高等学校计算机基础教育教材精选

# Python程序设计基础

鲁凌云　等编著

清华大学出版社

高等学校计算机基础教育教材精选

# Python程序设计基础

鲁凌云　等编著

清 华 大 学 出 版 社

北　京

## 内 容 简 介

本书是 Python 程序设计的入门书籍，将 Python 程序设计分为三大模块：面向过程的 Python 程序设计，面向对象的 Python 程序设计，以及 Python 程序设计综合实践题、模拟题和习题解析。面向过程的 Python 程序设计模块，介绍 Python 语言的入门基础知识，主要包括 Python 语言的技术起源、Python 语言的数据类型与表达式、Python 语言的流程控制语句，以及 Python 语言的函数设计方法。面向对象的 Python 程序设计模块，介绍 Python 的面向对象技术，包括继承、接口、封装的概念及实现，利用 turtle 库绘制图形，利用 NumPy 进行科学计算。Python 程序设计综合实践题、模拟题和习题解析模块，通过两个案例综合了前两个模块的重要知识点，通过两套模拟试卷夯实 Python 基础知识及核心技术，通过对每章的习题进行剖析，让读者尽快理解和掌握知识要点。本书所有案例都是基于 Python 3.0 版本以上的。

本书适合作为高等学校计算机通识类课程的教材，也可作为 Python 程序设计爱好者的入门书籍。



**图书在版编目(CIP)数据**

Python 程序设计基础/鲁凌云等编著. —北京：清华大学出版社，2019
(高等学校计算机基础教育教材精选)
ISBN 978-7-302-52418-2

Ⅰ. ①P…　Ⅱ. ①鲁…　Ⅲ. ①软件工具－程序设计－高等学校－教材　Ⅳ. ①TP311.561

中国版本图书馆 CIP 数据核字(2019)第 042149 号

**责任编辑**：龙启铭
**封面设计**：傅瑞学
**责任校对**：焦丽丽
**责任印制**：沈　露

**出版发行**：清华大学出版社
**网　　址**：http://www.tup.com.cn，http://www.wqbook.com
**地　　址**：北京清华大学学研大厦 A 座　　**邮　　编**：100084
**社 总 机**：010-62770175　　**邮　　购**：010-62786544
**投稿与读者服务**：010-62776969，c-service@tup.tsinghua.edu.cn
**质量反馈**：010-62772015，zhiliang@tup.tsinghua.edu.cn
**课件下载**：http://www.tup.com.cn，010-62795954
**印 装 者**：北京嘉实印刷有限公司
**经　　销**：全国新华书店
**开　　本**：185mm×260mm　　**印　　张**：13.75　　**字　　数**：321 千字
**版　　次**：2019 年 5 月第 1 版　　**印　　次**：2019 年 5 月第 1 次印刷
**定　　价**：39.00 元

产品编号：064241-01

# 前言

Python 语言以其“语法简单、句式清晰、高效实现”等特点逐渐成为当今世界最流行的编程语言之一。随着我国不断加强人工智能(Artificial Intelligence,AI)发展战略,Python 语言已被广泛应用于人工智能产品的研发、行业大数据分析等各个领域。掌握必要的 Python 语言已成为新世纪人才具备的基础素质之一。本书是 Python 语言的入门教材,期望能够为初学者打下良好基础,为初学者开启一扇探索 Python 语言与行业有效结合的大门。本书具有以下特点。

## 1. 定位准确

本书主要是为非计算机专业学生进行 Python 程序设计学习而编写的,考虑到这部分学生的程序设计基础比较薄弱,因此,本书的学习目标主要是将程序设计与本专业相结合,通过大量示例讲述程序设计语言中的奥妙。

## 2. 注重实践

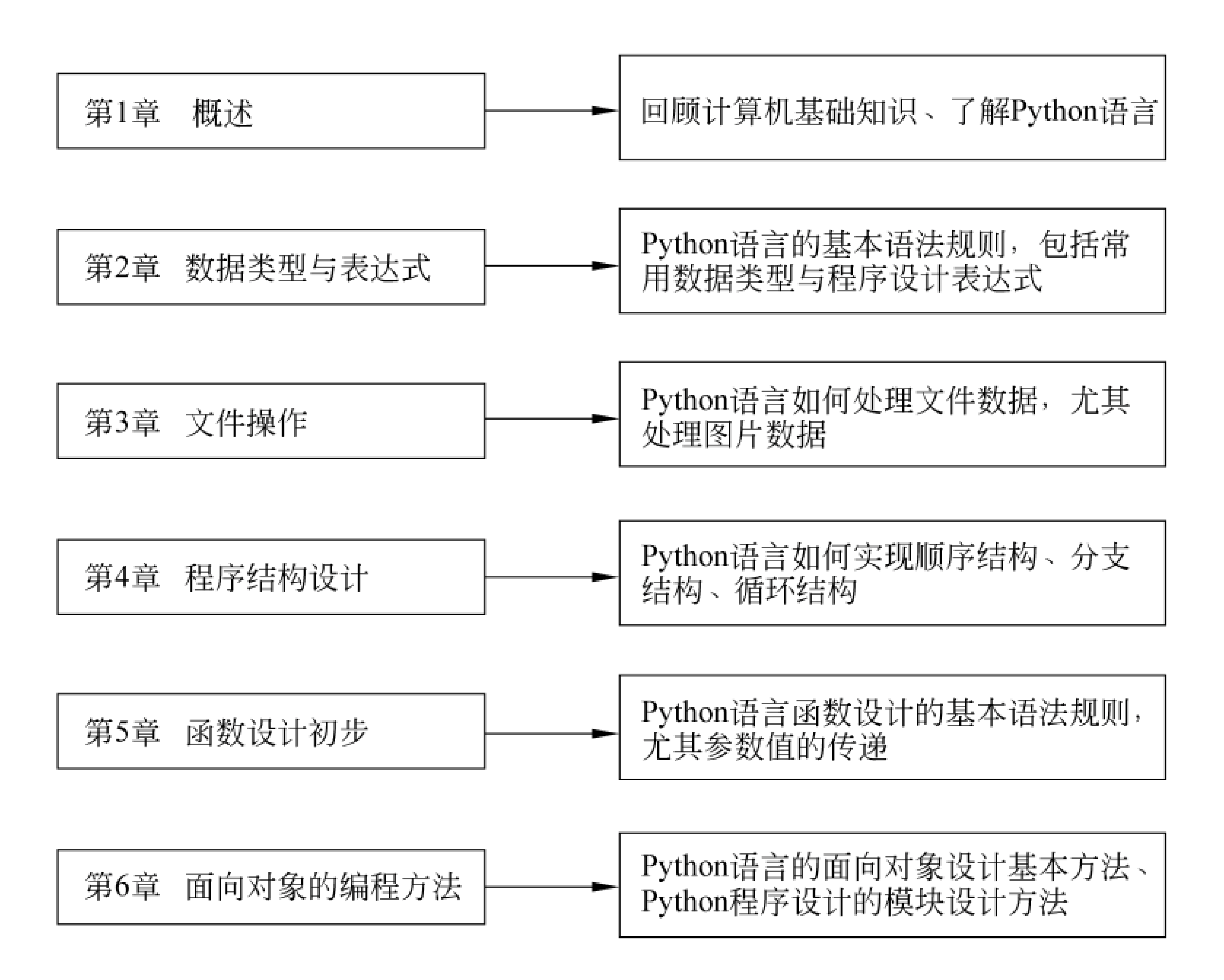
对非计算机专业的学生,在学习了必要的程序设计语法和规则后,更关注的是 Python 语言的实际应用,因此本书注重介绍使用 Python 语言编写程序来解决专业中可能遇到的工程问题。

## 3. 便于自学

本书由浅入深,通过大量的示例讲解,便于学生根据示例的提示,独立完成 Python 语言程序的编写与调试,有助于培养学生独立解决问题的能力,据此激发学生与本专业知识相结合的创新能力。

本书的篇章结构如下图所示。

本书由北京交通大学计算机学院鲁凌云任主编,诸强任副主编,张英俊、王瑞平两位教师负责了部分章节的编写。同时,也特别感谢北京交通大学熊柯教授、高博副教授、北京服装学院刘正东教授参与书稿的编写并提出宝贵建议。其中,鲁凌云负责编写第 1 章和第 2 章,诸强负责编写第 3 章和第 5 章,张英俊负责编写第 4 章,王瑞平、刘正东负责编写第 6 章。




与本书配套的教学资源，可登录“中国大学 MOOC(爱课程)”网站获取，链接如下：

https://www.icourse163.org/course/NJTU-1003368009

由于本书是为非计算机专业学生编写的教材，难易程度是否适合，还需要时间的检验，也欢迎广大读者对本书的各类错误批评指正。

编　者

2019 年 3 月

# 目录

# 第1章 概 述

## 1.1 计算机工作原理

### 1.1.1 计算机起源

中文名词“计算机”来源于英文 computer。英文单词 teacher(教师)、worker(工人)都是指人,透过 computer 英文单词的表面意思,还可以理解为“能够计算的人”。computer 这个词大约在 19 世纪 40 年代提出,也许那个时候的科学家有一种美好的愿景,希望有种机器能够模仿人的大脑,像人的大脑一样分析问题、处理问题。这里我们有必要先了解一下大脑的工作过程,例如,计算“8+6÷2=?”,如图 1-1 所示。人的眼睛看到这个数学表达式,经过分析,会通过嘴说出答案 11。首先,将这个表达式记在大脑中,再经过脑神经元的思考,结合数学知识,先算出“6÷2=3”这一中间结果,然后再给出“8+3=11”这一最终结果,11 这一结果将会保存在某个神经元。我们可以把这一结果通过声音告诉其他人,也可以写在纸上告诉别人,还可以记在脑海中不告诉其他人。通过做这一简单运算题,就可发现一条规律:首先,通过眼睛等感觉器官将捕捉的信息输送给大脑并存储起来,然后,对这一信息进行加工处理,最后,由大脑控制人把最终结果以某种方式表达出来。

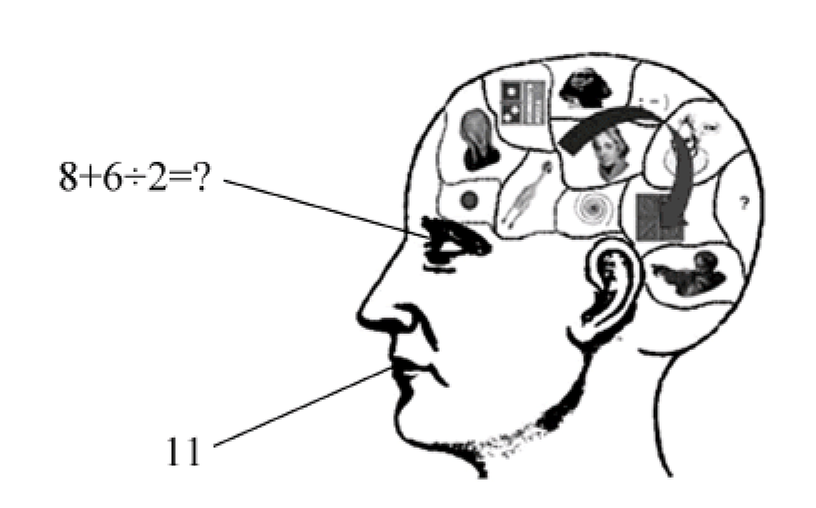


图 1-1 大脑的工作过程

是否可以设计一种机器来模拟或代替人脑的部分功能?计算机之父阿兰·图灵、控制论之父诺伯特·维纳和数字计算机鼻祖冯·诺依曼从不同的角度证明了这一假设的可能性与可行性。而且他们还从不同的视角预言,计算机可以模拟甚至超越人脑的某些能力(例如判断、推理、记忆等能力)。

阿兰·图灵是一名数学家(见图 1-2)。他发表了一篇经典论文《论可计算数及其在判定问题中的应用》。阿兰·图灵提出了“程序控制”思想,阐明了“有一种机器,也能像人脑一样执行指令序列”,并且给出了一种计算模型,即著名的图灵机(Turing Machine)模型。这种假想的机器由一个控制器和一个两端无限长的工作带组成,工作带被划分成一个个大小相同的方格,方格内记载着给定字母表上的符号。控制器带有读写头并能在工作带上按要求左右移动。随着控制器的移动,其上的读写头可读出方格上的符号,也能改

写方格上的符号。这种机器能进行多种运算，并可用于证明一些著名的定理。这就是最早给出的通用计算机模型。

诺伯特·维纳(见图 1-3)从控制与通信的角度论述计算机的实现问题。在《控制论》一书中，维纳认为计算机的设计应该遵循下述原则。



图 1-2　阿兰·图灵

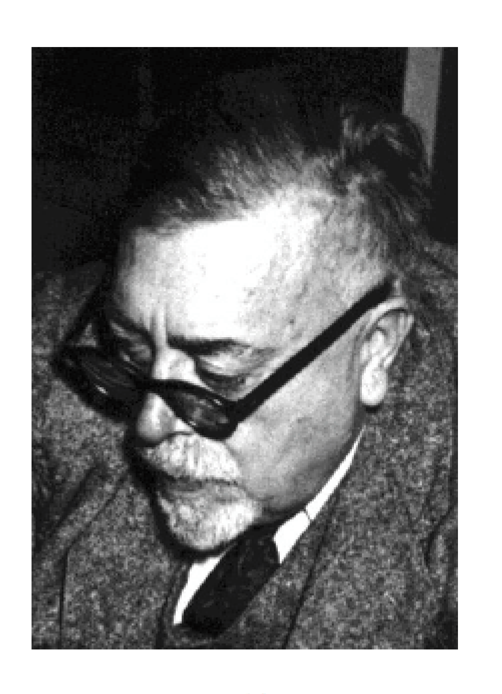


图 1-3　诺伯特·维纳

(1) 计算机的核心部分，加法和乘法装置应当是数字式的。

(2) 加法和乘法采用二进制，实现这种方式比十进制更经济，可以由电子管完成。

(3) 从把数据放入机器到取得结果，全部运算序列应自动进行，中间没有人的干预。

(4) 运行中涉及的逻辑判断应由机器自身完成。

(5) 机器应包括存储数据和控制命令的装置。

维纳建议美国政府研制数字计算机，得到采纳后，成立以冯·诺依曼负责的电子数字计算机研究课题组。1946 年，世界上第一台电子计算机 ENIAC 问世。今天的计算机也被称为冯·诺依曼计算机。当时，研究和开发 ENIAC 计算机的目的是为军事服务，主要是为了计算弹道和火力表。随着 ENIAC 的诞生，人类拉开了计算机发展的序幕。

冯·诺依曼在实现计算机的过程中，也汲取了图灵和维纳的思想。运算器采用了二进制方法实现，电子器件采用电子管。ENIAC 由 18800 个电子管组成(如图 1-4 所示)，利用电子管实现了计算机的核心部件 ALU，每秒钟可完成 5000 次加减法或 400 次乘法运算。





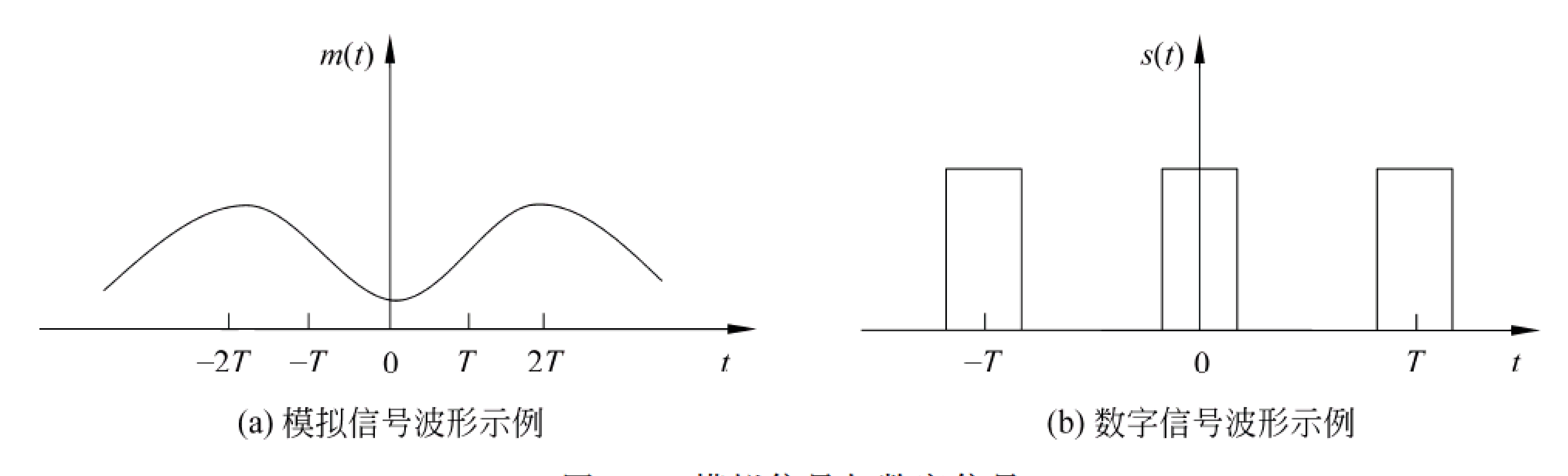
图 1-4　电子管与世界上第一台计算机 ENIAC

但是电子管体积大，耗电多，容易损坏，例如，ENIAC 耗电大约 150kW。因此，随着电子技术的不断发展，计算机的核心部件逐渐用晶体管、集成电路等电子元器件替代。从第一台计算机诞生至今，计算机都是由电子器件实现，因此，我们目前使用的绝大多数计算机称为“电子计算机”。

随着科技的突飞猛进，人类对计算机的认识不断深入，除了“电子计算机”，未来可能还会出现各种物理形式的计算机。目前，科研工作者正不断探索“量子计算机”“生物计算机”“光子计算机”等。

## 1.1.2 二进制数

计算机是电子设备，利用计算机可以处理数字信号的数据，也可以处理模拟信号的数据。数字信号与模拟信号的区别如图 1-5(a)和图 1-5 (b)所示。我们不难发现，数字信号的最大特点是容易实现，只要我们事先设定一个阈值，大于这个阈值，我们认为这个信号表示“1”，否则认为是“0”。利用电子元器件较容易实现数字计算机，而实现模拟计算机需要较高精确度的电路结构，是比较困难的，因此“模拟电子计算机”在计算机的发展史上稍纵即逝，很快被淘汰。我们目前使用的绝大多数计算机都是“数字电子计算机”。



(a) 模拟信号波形示例　　(b) 数字信号波形示例

图 1-5　模拟信号与数字信号

数字计算机的电路只有两种工作状态，即开或关。为了表示这两种工作状态，常常用 1 表示“开”，用 0 表示“关”，即计算机硬件用一系列的高低电平表示“0”或“1”。在计算机内部，由于只能处理“0”或“1”组合的信息，因此无论是用户数据，还是控制这些数据的“命令”，都需要用二进制数表示。

二进制数只有两个数字“0”或“1”，按照“逢二进一”的原则计数，即每位计满 2 时向高位进 1。例如，二进制的数值“100”对应十进制的数值“4”，十进制的数值“10”对应二进制的数值“1010”。一连串的二进制数，例如“1000110010001100”，不但可以表示数值，也可以表示计算符号、图片的像素。那么，计算机存储二进制数的最小单位就是“比特”(bit)，简写为 b。字节(Byte)是计算机处理数据的基本单位，简写为 B。比特与字节之间的关系如图 1-6 所示。

计算机利用“字节”的方式存储指令和数据。计算机“指令”就是指挥计算机工作的命令，一系列按一定顺序排列的指令就构成了程序。数据好像是“士兵”，而指令更像是“指挥官”。数据的一举一动，都要服从“指挥官”的命令。不同的指令占用的字节数不同；不

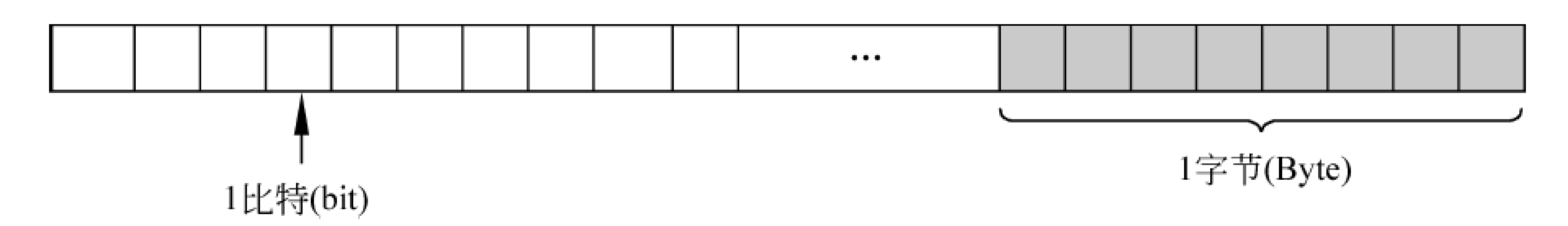



图 1-6　比特与字节之间的关系示意图

同类型的数据，所占的字节数也不同。例如，对于 Python 指令“a＝3”，其中“3”是数据，计算机内部将会分配 2 字节存储整数 3；对于 Python 指令“a＝'C'”，其中“C”是数据，计算机内部将会分配 1 字节存储字符 C；对于 Python 指令“a＝3.58”，其中“3.58”是数据，计算机内部可能会分配 4 字节存储小数 3.58，如图 1-7(a)～图 1-7(c)所示。给变量分配的字节数目的大小，与编译器、操作系统等编译环境有关，不同的编译环境可能会为同一数据类型分配不同的字节数。

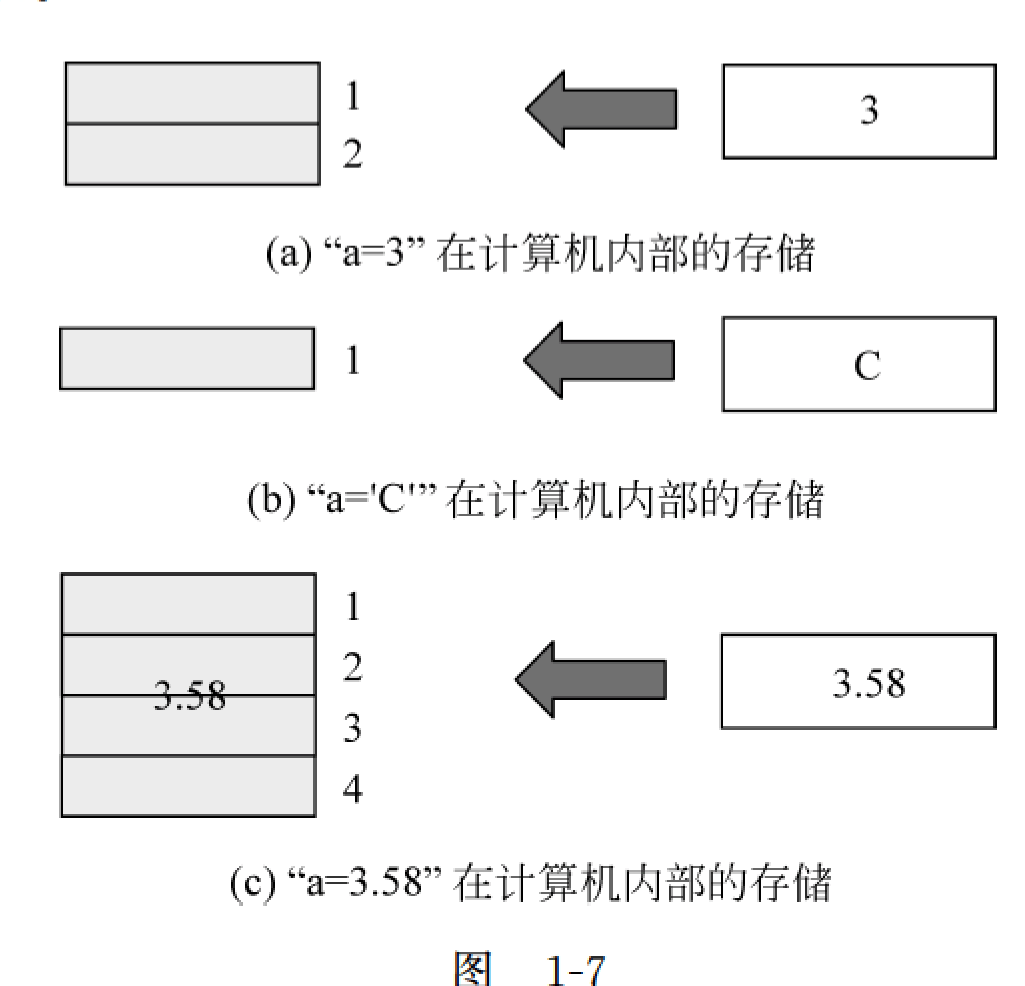



图　1-7

## 1.1.3　计算机内部结构

计算机是如何利用存储的指令和数据工作的？这就涉及计算机内部结构。目前，大多数的计算机结构延续的还是冯・诺依曼设计的结构。冯・诺依曼（如图 1-8）提出了“将计算机要处理的程序和数据先放在存储器中，在计算机的运算过程中，由存储器按事先编好的程序，快速地提供给微处理器进行处理，在处理过程中不需要用户干预”的原理，而计算机之所以能够高效工作，就是基于存储程序和程序控制这个原理。



图 1-8　冯・诺依曼

计算机主要包括五部分：输入设备、输出设备、存储器、运算器、控制器，如图 1-9 所示。输入设备就像人的眼睛和耳朵一样，负责传递各类数据。输出设备就像人的嘴一样，负责传递大脑思考分析后的数据。存储器、运算器和控制器配合起来工作，就像人的大脑一样，可以分析、判断问题。运算器在处理问题时，如果

问题较复杂,需要一条条分析、推理或是判断。控制器负责发送各类存取数据的命令。我们不难发现,“数据什么时候进入计算机”“什么时候数据被处理”“什么时候把处理结果显示出来”,所有操作都与计算机的控制器密切相关。冯·诺依曼设计计算机的过程中,把“运算器”“控制器”“寄存器”封装成计算机的核心器件,即中央处理器(Central Processing Unit,CPU)。

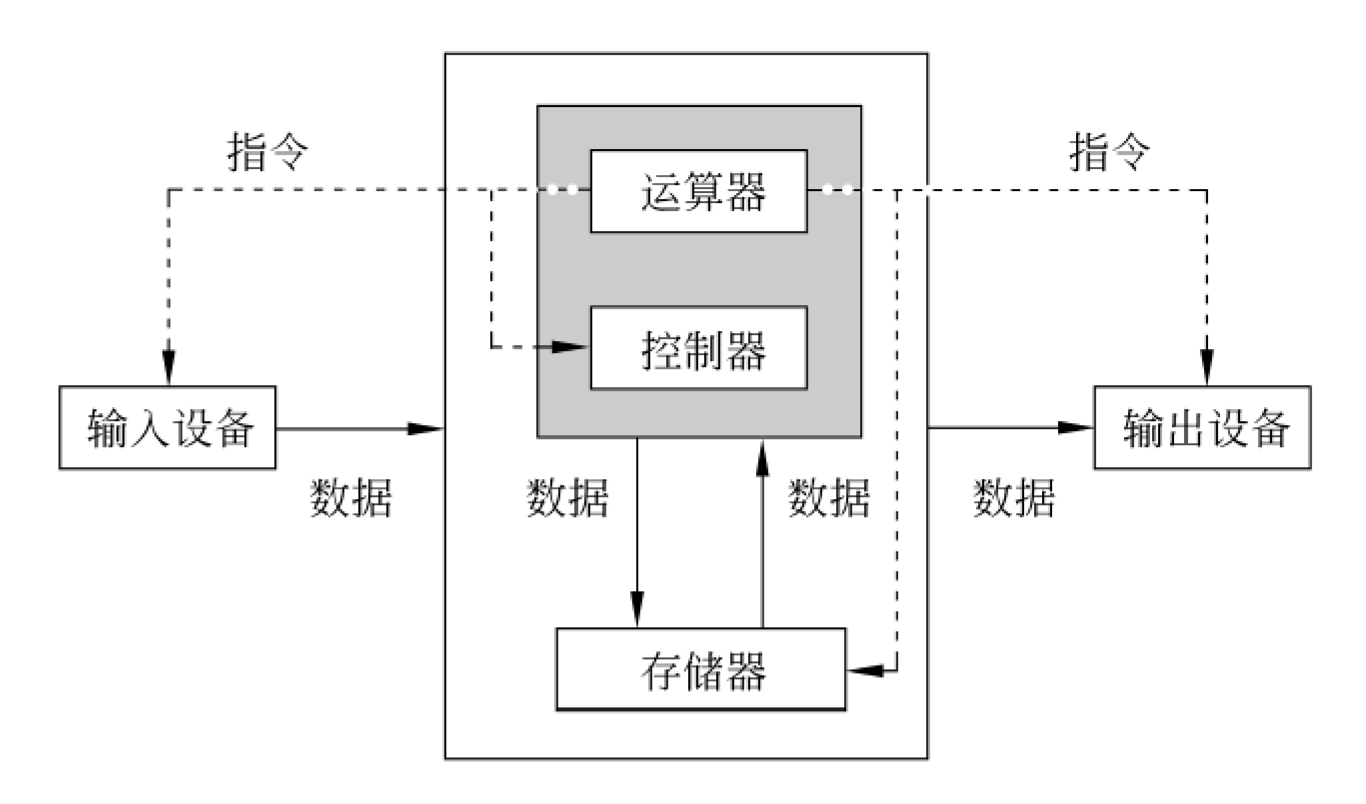



图 1-9　冯·诺依曼体系结构

## 1. 运算器

运算器是任何计算机的核心设备之一,其作用就是用来进行算术运算和逻辑运算,是计算机的主体。在控制器的命令下,运算器接收要运算的数据,完成程序指令指定的算术运算或逻辑运算。

## 2. 控制器

控制器是计算机的指令控制中心,用来分析指令、协调 I/O 操作和内存访问。控制器从存储器中逐条取出指令、分析指令,然后根据指令要求完成相应操作,产生一系列控制命令,使计算机各部分自动、连续并协调地工作,作为一个有机的整体,实现数据和程序的输入、运算并输出结果。

## 3. 存储器

存储器是用来存储程序、数据、运算的中间结果及最后结果的设备,计算机中的各种信息都要存放在存储设备中。根据存储设备在计算机中处于的不同位置,可分为主存储器(也称为内存储器,简称为内存)和辅助存储器(也称为外存储器,简称为外存)。从存储介质构成原理角度,可分为磁表面存储器、半导体存储器、光介质和磁光介质存储器等。

在计算机内部,直接与 CPU 交换信息的存储器称为内存,用于存放计算机运行期间所需的信息,如指令、数据等。外存是内存的延伸,其主要作用是长期存放计算机工作时所需要的系统文件、应用程序、用户程序、文档和数据等。当 CPU 需要执行某部分程序和数据时,由外存调入内存以供 CPU 访问。可见,外存用于长期保存数据和扩大存储系统容量,主要有磁盘、磁带或 U 盘。由于 U 盘具有存储容量大、价格低廉、性能好等特点,

已成为目前微型计算机最常用的移动存储设备。

如果说CPU是计算机非常重要的一个部件，那么存储器就应该是计算机特别重要的一个部件。输入到计算机的数据放在什么地方？处理这些数据的命令被放在哪里？数据处理后的结果又放在什么地方？这些问题都需要在计算机设计过程中认真考虑。存储器的任务艰巨，工程师们把存储器分成了三种不同的类型，承担着不同的存储任务。一种是暂时存储器，常常称为计算机的内存；一种是CPU内部的存储器，称为寄存器(register)；一种是长期保存数据和命令的地方，称为外存。当计算机关机时，内存不会长期保存数据，因此需要把数据存放在外存中。

### 4. 输入设备

输入设备是用来完成数据输入功能的部件，即向计算机输送程序、数据以及各种信息的设备。常用的输入设备有键盘、鼠标、扫描仪、U盘、磁盘和触摸屏等。

### 5. 输出设备

输出设备是用来长期保存数据的，即将计算机工作的中间结果或最终的处理结果从内存传送到外存的设备。常用的输出设备有显示器、打印机、绘图仪、U盘以及磁盘等。

# 1.2 计算机语言

## 1.2.1 指令

指令是指挥计算机完成某个操作的命令，发出的指令要能被计算机的输入设备、输出设备、存储器、运算器、控制器理解并执行。设计好的计算机硬件是为了更好地执行"指令"。为解决某个问题而设计的一系列有序指令的集合称为程序，而程序是用某种"计算机语言"描述的。计算机的工作过程就是根据程序处理数据，得到结果。若干段程序和相关的文档集合就构成了软件。软件可实现比较完整的一个功能，如图1-10所示。

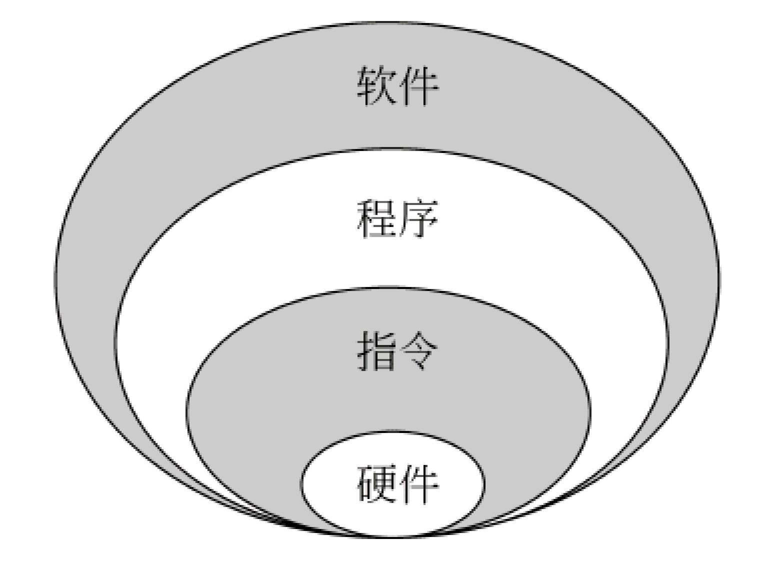


图1-10 计算机软件层次结构

## 1.2.2 计算机语言发展史简介

计算机语言是人与计算机之间交流的语言。第一台计算机ENIAC直接识别并执行人类编写的指令，人类使用"机器语言"与ENIAC进行交流。机器语言很难掌握，不具备通用性，而且，机器语言是整个计算机体系中最深奥的接口，是硬件和软件直接的中间件。

汇编语言的出现大大提高了人们控制计算机的热情，利用汇编语言可以解决很多问题。汇编是一种利用一些特定助记符表示指令的语言。与机器语言相比，汇编语言比较

容易掌握。但计算机的硬件只能处理二进制数,汇编语言最终要转换成二进制。“0”和“1”的世界,人类是不熟悉的,因此需要编译或者解释工作。我们把汇编语言翻译成机器语言的过程称为编译。但是,只有程序员对计算机的工作流程很熟悉,才能使用汇编语言编写程序,对程序员的要求较高,从某种程度上也限制了计算机的普及和应用。

为了让更多的人更容易控制计算机,产生了高级的计算机语言。伴随着人类进步和计算机技术的进一步发展,计算机的各类高级语言正帮助人们解决各式各样的问题。高级语言是一种更接近人们对求解过程或问题描述的计算机语言。这种语言允许用英文编写程序,程序中所使用的运算符号或者表达式,都与我们日常用的数学表达式十分相似。高级语言容易学习,通用性强,便于推广和交流,是很理想的一种程序设计语言。高级语言发展于 20 世纪 50 年代中叶到 20 世纪 70 年代,有些流行的高级语言已经被大多数计算机厂家采用,固化在计算机的内存里。例如,C、FORTRAN、C++ 、Java、Python 等都属于高级语言。相对于高级语言,机器语言和汇编语言又称为低级语言。

### 1.2.3 程序设计中的“变量”与“变量值”

利用计算机的某种语言可以实现某种功能,称为“程序设计”。程序设计的目的是帮助处理数据,并得到理想的结果。那么,输入的数据是千变万化的,就像一道数学题“两个数相加的得数是多少?”可以设计为表达式“Z=X+Y”,未知数 X 和 Y 可以填写很多种可能的数据。根据 X 和 Y 的值,可以得到 Z 值。计算机中的变量(variable)是一个抽象概念,可以理解为允许存放数据的空间。计算机程序设计中的表达式“Z=X+Y”涉及 3 个变量,即计算机要准备“3 块空间”来存放数据。

我们这个世界,从沙粒到太阳,从原生生物到人,都处于永恒的产生和消灭中,处于不断的流动中,处于无休止的运动和变化中。当然,这其中包括了计算机,计算机如何“流动”,很大程度依靠计算机指令。在计算机的内部,如果要存储一些数据,就要准备存放数据的空间。当声明一个变量,计算机就会分配空间来存放数据。变量可以通过“变量名”访问。在表达式“Z=X+Y”中,Z、X、Y 为变量名。放在变量里面的数据可以是数值、字符或一串字符等。

变量里面能放什么值,还要看计算机语言的规则。例如,计算机的 C 语言要求在使用变量前,必须先声明变量类型,也就是说,类型不同,给变量分配的存储空间也不同。但 Python 语言很简单,用户把什么类型的数据存入计算机的空间,变量就是什么类型的。在程序中使用变量,一般需要三个步骤。

(1) 声明变量:就是给这块能够存放变量值的存储空间起名。但是划分多大空间需要看存储值的类型,即数据类型。在 C 语言中,一定要先声明变量,但 Python 却可以省略了这一步。这是 Python 语言与 C 语言的不同之处,Python 不需要声明变量的类型。C 语言是事先为变量定义好一个数据类型,根据这个数据类型的特点分配存储空间。例如,Python 程序中如果定义 a=5,那么 a 就是整型数值;如果定义 a=4.5,那么 a 就是浮点型数值;如果定义 a='Beijing',那么 a 就是字符串类型数值。

(2) 变量赋值:使用赋值符号“=”,相当于从右到左的标识符“←”,也可以这样理

解，把右侧的数值给左侧的变量。例如，a=3 就是把整数 3 存放在变量 a 中。还可以利用函数给变量赋值，例如，a=input()把从键盘输入的字符存放在变量 a 中。这里，input()是一个标准函数。

(3) 使用变量：计算机中存放了数据，利用设计的程序使用和控制数据。例如，对于程序语句 a=3 和 a=10−a，a 的初始值是 3，运行语句 a=10−a 后，变量 a 的数值为 7。因此，也可以理解为，变量值可根据不同的控制语句发生改变。

**【例 1-1】** 一个罐子装了 10 颗糖，每天吃 1 颗糖。3 天以后，糖的数目是多少？描述计算机处理这个问题的过程如图 1-11 所示。

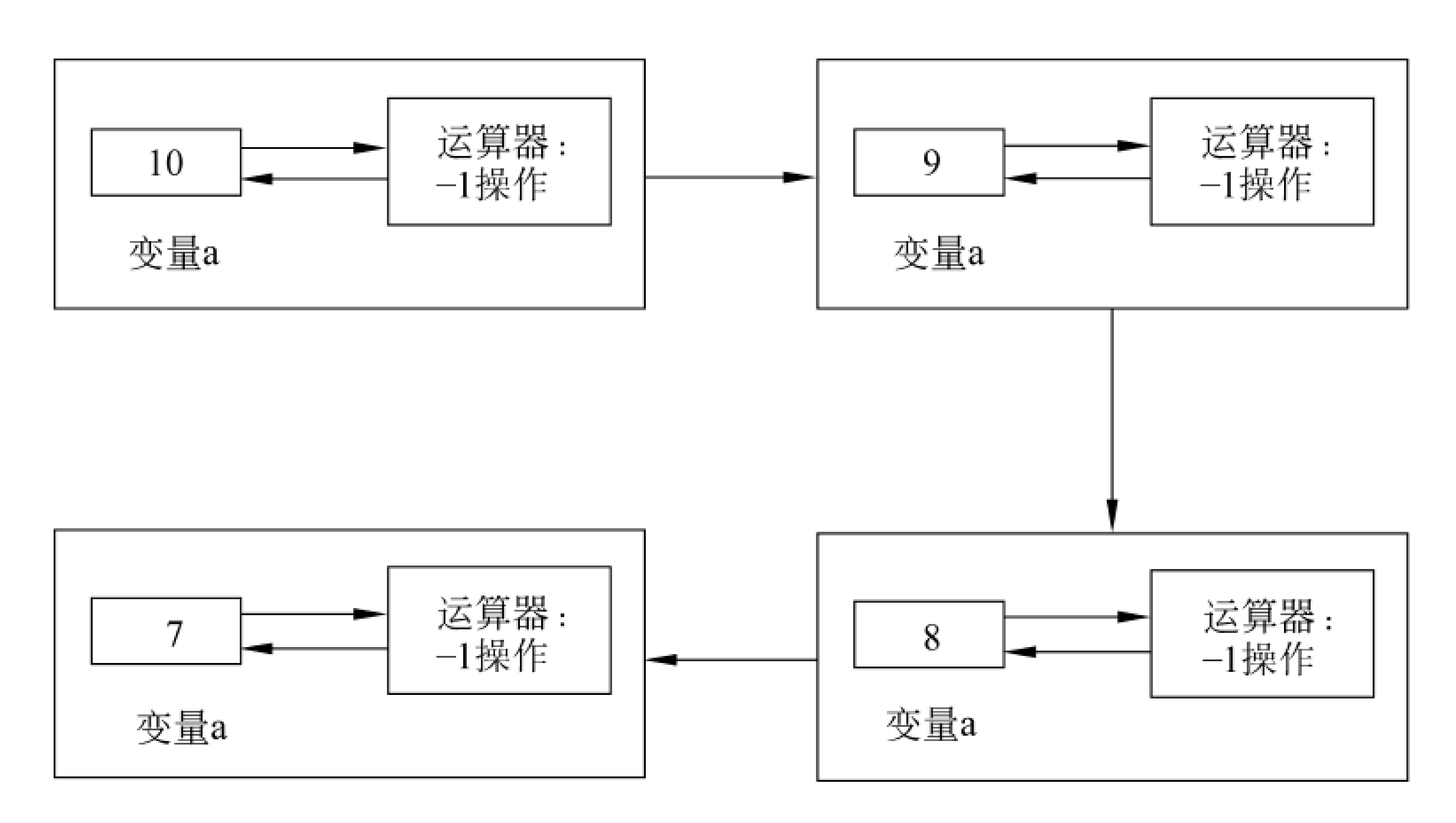



图 1-11　变量的存储值变化情况

**【分析】**

① 假设一个罐子相当于变量 a，运行表达式 a=a−1。最开始存放一个整数 10，第一天，减 1 后，将结果 9 仍然存放在变量 a 中。第二天，9 减 1 后，将结果 8 仍然存放在变量 a 中。第三天，8 减 1 后，将结果 7 仍然存放在变量 a 中。

② 最后变量 a 内存中存放的数值是 7。

## 1.3　Python 语言

Python 语言属于计算机高级语言，它功能强大，对程序设计初学者来说易于掌握。Python 语言是由荷兰人 Guido Van Rossum 于 1989 年创造。他用英国喜剧 *Monty Python and the Flying Circus* 中的“Python”命名了这门语言。Python 作为一门程序设计语言主要具有如下特点。

### 1. Python 语言能够减少代码量

Python 语言是当今世界最灵活和易用的编程语言。Python 可减少很多编写、调试和维护工作。使用 Python 进行程序开发，可以数倍地提升编码效率，不到 C、C++ 或 Java 一半的代码行数，将大幅度减少开发过程和维护阶段的工作量。与其他语言比较，

Python 支持“大规模编程”,使其适用于开发大型系统。

### 2. Python 语言的兼容性

由于 Python 是开源的,也就意味着它不可能被某一个公司所掌控。总之,Python 往往是 C、C++ 或 Java 等系统开发语言的一个不错的替代品。但 Python 的执行速度并不如 C 语言等。因此,在大数据、云计算等领域解决求逆矩阵、向量相似度等问题时,Python 并不具有良好的运行效率,让 Python 去做这些必然会造成计算机性能下降。然而,Python 的集成机制可以轻松地连接使用 C、C++ 或 Java 语言编写的模块。可以根据情况需要,使用 Python 来做框架,在核心 CPU 密集操作部分调用 C 或者其他高效语言,这样,开发效率和性能都得到保障。因此,Python 又称为“胶水语言”。

### 3. Python 语言结合了解释和编译

计算机硬件是不能识别高级语言的,所以当运行一个高级语言程序的时候,就需要一个“翻译机”来负责把高级语言转变成计算机能读懂的机器语言。这个转变过程分成两类,第一种是编译,第二种是解释。这两个过程需要配置“编译器”或“解释器”来完成。编译器是把源程序的每一条语句都编译成机器语言,并保存成二进制文件。也就是说,编译型语言在程序执行之前,会先通过编译器对程序执行一个编译的过程,把程序转变成机器语言。这样计算机可以直接运行机器语言程序,速度很快。最典型的例子就是 C 语言。而解释器是在执行程序时,一条一条地解释成机器语言并给计算机执行,运行速度不如编译的程序快。高级语言 Java 的编译比较特殊,没有直接编译成机器语言,而是转换成“字节码”,在虚拟机上用解释器来执行字节码。使用 Python 语言设计的程序采用了类似的过程。Python 会先将源代码.py 文件编译成中间形式的字节码(Bytecode)并存放在内存当中,然后在真正执行时,将字节码解释为机器可识别的二进制码。

## 1.4 第一个 Python 程序

**【例 1-2】** 从键盘输入一个数,让这个数乘以 2 再加 10 后,在屏幕上显示计算结果。

**【分析】**

① 完成这个任务,描述如图 1-12 所示。

② 如何设计一条 Python 语句,让计算机接收用户输入的数字?

开发 Python 语言的工程师已经设计好特殊的语句,用户可以直接使用。程序设计过程中,把这些设计好的语句称为“标准函数”。例如,Python 常用的输入数据的函数有标准函数 input()。

③ 输入的数放在计算机什么地方?

计算机采用了“变量”的方法来存放数据值。变量的值不但可以是数字,还可以是字符。变量是计算机编程中的一个重要概念。简单地说,可使用变量来存储任何数据。变量可以随着程序的运行而改变其表示的值。

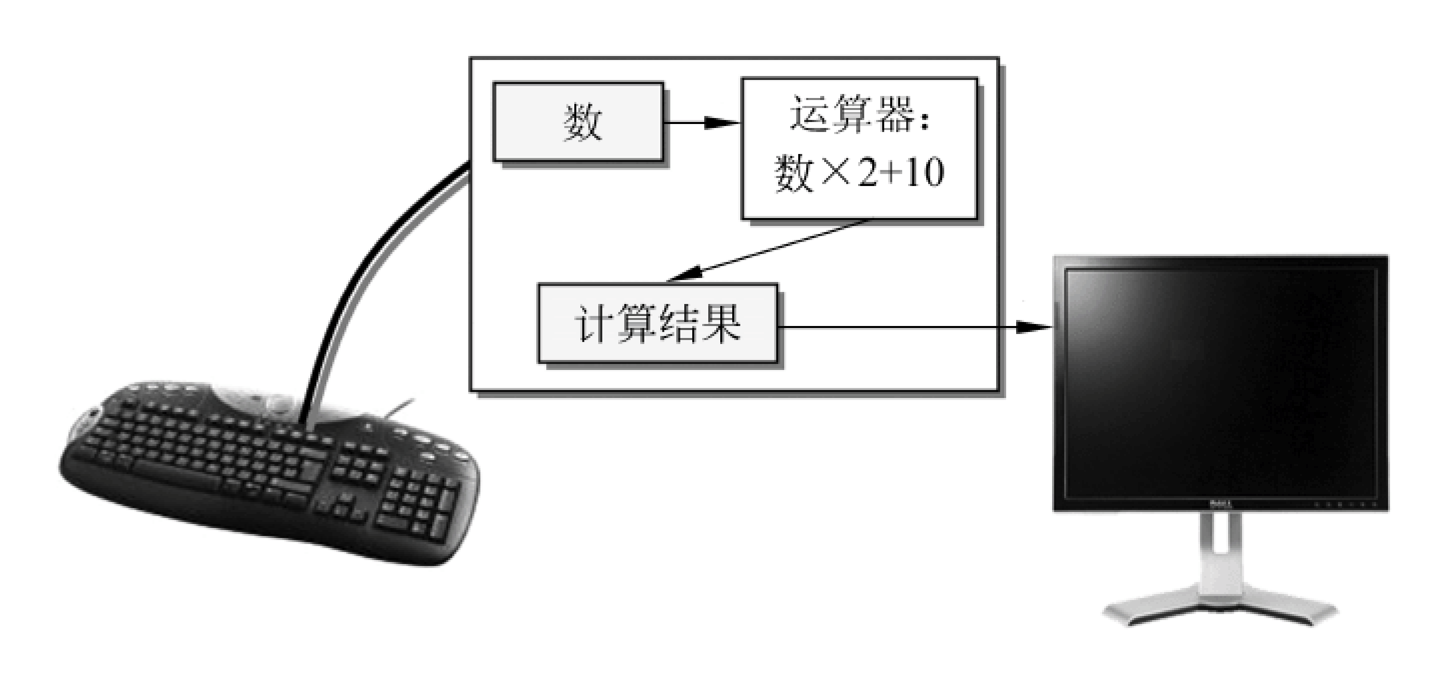



图 1-12　计算机处理数据的过程

④ 计算机将运行的结果显示在什么地方?

显示在计算机输出设备屏幕上。Python 提供了标准的输出函数 print()。

代码:

```
number=input("请输入一个数字:")
result=int(number)*2+10
print("结果是:",result)
```

运行结果:

```
请输入一个数字:34
结果是:78
```

# 本 章 小 结

(1) 了解计算机的发展历史。

(2) 掌握计算机的基本组成原理。

(3) 熟练掌握 Python 的安装方法。

(4) 正确理解变量的概念以及变量赋值。

(5) 掌握 Python 程序的基本编写思路。

# 习　　题

## 1. 简答题

(1) 列举几位对计算机发展产生过重要影响的人物,并简述他们的贡献。

(2) 简述计算机采用二进制的原因。

(3) 什么是摩尔定律？它描述的是什么现象？

(4) 有 15 瓶药水，其中一瓶有毒。假定一只小白鼠喝下药水后是否中毒需要一个小时才会体现出来。如果有 4 只小白鼠，是否有办法用一个小时的时间检测出有毒的药水？

## 2. 填空题

(1) 基于冯·诺依曼思想而设计的计算机硬件由运算器、________、________、________和输出设备 5 部分组成。

(2) 一个字节等于________位。

(3) 世界上首台数字计算机诞生于________年。

(4) 程序设计语言的三个大类分别是________、________、________。

## 3. 选择题

(1) 世界上第一台电子计算机是【　　】。

A) ENIAC　　B) ABC　　C) EDVAC　　D) Mark Ⅱ

(2) 最早计算机主要是用于【　　】。

A) 数据处理　　B) 科学计算　　C) 辅助设计　　D) 过程控制

(3) 以下不同进制的四个数中，【　　】是最小的数。

A) $(101101)_2$　　B) $(52)_8$　　C) $(2B)_{16}$　　D) $(46)_{10}$

(4) 设一个具有 20 位地址和 64 位字长的存储器，该存储器能存储【　　】个字节的信息。

A) 8MB　　B) 4MB　　C) 2MB　　D) 1MB

(5) Python 是由【　　】发展来的。

A) C 语言　　B) ABC　　C) FORTRAN　　D) Pascal

(6) 英国科学家巴贝奇提出一种通用的计算机设计思想，称为【　　】。

A) 加法器　　B) 微机　　C) 差分机　　D) 分析机

(7) 物理器件采用晶体管的计算机被称为【　　】计算机。

A) 第 1 代　　B) 第 2 代　　C) 第 3 代　　D) 第 4 代

(8) 摩尔定律指出芯片上集成的晶体管数目每【　　】个月翻一番。

A) 6　　B) 12　　C) 18　　D) 24

(9) 下列不属于新型计算机的是【　　】。

A) 超导计算机　　B) 量子计算机　　C) 半导体计算机　　D) DNA 计算机

(10) 下列语言中【　　】不是面向对象程序设计语言。

A) Java　　B) C 语言　　C) Python　　D) C#

(11) 程序中的错误主要分为语法错误和【　　】。

A) 逻辑错误　　B) 系统错误　　C) 自定义错误　　D) 结构错误

(12）程序可以不满足算法的【　　】性质。

A）有外部量作为输入　　B）产生至少一个输出

C）指令清晰无歧义　　D）指令执行次数有限

(13）下列选项中，属于非通用编程语言的是【　　】。

A）C 语言　　B）SQL　　C）Java　　D）Python

# 第2章 数据类型与表达式

## 2.1 数据类型

我们身处大数据的时代,各种形形色色的数据充斥着这个世界。通常我们会把数据进行分类整理。同一类数据的全体称为一种数据类型。在程序设计高级语言中,数据类型用来说明一个数据在数据分类中的归属,它是数据的一种属性。这个属性限定了该数据的变化范围。为了程序设计的需要,根据数据结构的种类,每种高级语言定义了一系列的数据类型。不同的高级语言所定义的数据类型不尽相同。表 2-1 为 Python 语言常用数据类型及其示例。

**表 2-1 Python 语言常用数据类型**

| 序号 | 数据类型 | | 示例 |
|---|---|---|---|
| ① | 数值类型 | 整型 | a=1234 |
| | | 浮点型 | b=3.1415 |
| | | 复数 | d=3+4j |
| ② | 字符串类型 | 单字符型 | a='A' |
| | | 字符串型 | b='spam' |
| ③ | 布尔类型 | | a=True<br>b=False |
| ④ | 列表类型 | 单一类型列表 | a=[1,2,3,4] |
| | | 混合类型列表 | b=[1,'a',2,'there'] |
| | | 嵌套式列表 | c=[1,[2,'there'],4] |
| ⑤ | 字典类型 | | a={'food':'spam', 'taste':'yum'} |
| ⑥ | 元组类型 | 单一类型元组 | a=(1,2,3,4) |
| | | 混合类型元组 | b=(1,['a',2],'there') |
| | | 嵌套式元组 | c=(1,(2,'there'),4) |
| ⑦ | 文件类型 | | Myfile=open('data.txt','r') |

### 2.1.1　数值类型

客观世界中，我们常常用数值表示量的多少。例如，坚持跑步 35 天，某个物品的重量是 3.67 千克，“35”“3.67”就是数值。除了实数，在信号分析、量子力学还涉及虚数。如果这些数值作为数据需要计算机分析处理，在计算机中如何表示这些数值？Python 语言程序设计中，将这些表示数值的数据称为“数值类型”。Python 语言提供了三种数值类型，即整数类型、浮点数类型、有虚部的复数类型。

#### 1. 整数类型

根据实际需要，计算机需要为某些整数分配较大的内存空间，有的只需要较小的内存空间。据此，我们将整数类型分为标准整型和长整型。长整型能表示的数值范围是无限的，只与机器的内存空间有关。因此，Python 能够计算整数的较大阶乘。

（1）标准整型。

标准整型数值不但可以用我们熟悉的十进制表示，Python 语言规定还可以根据需要使用二进制、八进制、十六进制表示整数。Python 程序设计中，一般默认为十进制整型数值。如何区分某个数值是用哪种进制表示？除了十进制，其他进制都需要在数值前加上引导符号。二进制的引导符号为 0B 或者 0b；八进制的引导符号为 0O 或者 0o；十六进制的引导符号为 0X 或者 0x。

**【例 2-1】** 分别用十进制、二进制、八进制、十六进制表示整数 20。

```
A1=20
A2=0B10100
A3=0O24
A4=0X14
print(A1,A2,A3,A4)
```

运行结果：

```
20 20 20 20
```

分析例 2-1，A2、A3、A4 分别用不同的进制数表示十进制整数 20。其中，二进制“10100”、八进制“24”、十六进制“14”用不同的方式表示 20。如何获得“10100”“24”“14”这些不同进制的数值？这是一个将任何十进制数转换为其他进制数的问题。Python 语言为转换成其他进制数值提供了标准函数。bin()用于将十进制的数转换成二进制；oct()用于将十进制的数转换成八进制；hex()用于将十进制的数转换成十六进制。

```
A1=20
B2=bin(20)
B3=oct(20)
```

```
B4=hex(20)
print(A1,B2,B3,B4)
```

运行结果：

```
20 0b10100 0o24 0x14
```

整数可以进行各种常用的数值运算，例如，加法、减法、乘法、除法、求幂运算，如例 2-2 所示。

**【例 2-2】** 整数可以参与各种表达式运算。

```
a=15
b=15*4+20-10
c=15/3
print (a,b,c)
```

运行结果：

```
15 70 5.0
```

（2）长整型。

与 C 语言不同，Python 语言的长整数不限制数值的大小，只是与计算机内存限度有关。在使用过程中，如何区分标准整型与长整型数值呢？在 Python 2. X 中，通常的做法是在数字尾部加上一个大写字母“L”或小写字母“l”表示长整型，如例 2-2 所示。需要注意的是，在 Python 3. X 中不再区分标准整型与长整型，如果数字尾部加上“L”或“l”会报错。

**【例 2-3】** 表示长整型数值（只在 Python 2. X 中测试）并进行表达式运算。

```
a=3457475L
b=45555000L
c=a+b
print (a,b,c)
```

运行结果：

```
3457475 45555000 49012475
```

长整型可以保存较大取值范围的数值。例如计算幂运算，如例 2-4 所示。

**【例 2-4】** 求 2 的 100 次幂。

```
a=2
b=2**100
print (b)
```

运行结果：

```
1267650600228229401496703205376
```

## 2. 浮点数类型

计算机中的浮点数用来处理实数，即带有小数的数值。“1.0”和“1”在计算机中的表示方法不同，即“1.0”为浮点数。Python 语言规定了浮点数有两种表示形式，一种是十进制数表示法，由数字和小数点构成。需要注意的是，这里的小数点是不可或缺的，如“5.31”“531.0”“0.0”；另一种是指数表示法或者科学计数法，如“2018e2”或 2018E2 表示的都是 $2018\times10^2$，字母 e(或 E)之前必须有数字，字母 e(或 E)之后可以有正负号，表示指数的符号，如果没有符号，则默认为正号。指数必须为整数。

**【例 2-5】** 分别用十进制和指数方法表示浮点数 2018.8。

```
a=2018.8
b=2.0188e3
print (a,b)
```

运行结果：

```
2018.8   2018.8
```

需要注意的是，浮点数的取值范围有限制。浮点数最多可以输出 17 个数字，但浮点数只能保证 15 个数字是精确的，超过 15 位数字就会产生误差。因此，在 Python 程序设计中，采用整型数计算往往比采用浮点数计算可以获得更高精度的计算结果。分析例 2-6，我们发现，整数运算可以获得更大范围的数值，而浮点数至多能输出 17 位，而且超过 15 位后的数值已经不精确。

**【例 2-6】** 计算 2018201820182018×2018 和 2.018201820182018×2018，分析结果。

```
a=2018201820182018* 2018
b=2.018201820182018* 2018
print (a,b)
```

运行结果：

```
4072731273127312324   4072.7312731273128
```

## 3. 复数类型

复数由实数部分和虚数部分组成，一般形式为 x+yj，其中 x 是复数的实数部分，y 是复数的虚数部分，这里的 x 和 y 都是实数。注意，虚数部分的字母 j 大小写都可以，

如 5.6+3.1j 与 5.6+3.1J 是等价的。

**【例 2-7】** 利用 Python 提取复数 1.5+0.5j 的实部和虚部。

```
a=1.5+0.5j
b=a.real
c=a.imag
print (a,b,c)
```

运行结果：

```
(1.5+0.5j) 1.5 0.5
```

### 2.1.2 字符串类型

字符是计算机用编码方式存储的字母、数字、汉字、符号等，如利用 ASCII 编码存储 1、2、3、A、B、C、!、#、%、*、+等字符。需要注意的是，字符“1”和数值 1 是不同的。我们可以用字符“0010”表示房间号，但是“0010”不是数值。如何区分字符“1”和数值 1？我们利用一对单引号或者双引号。如果数字加上一对单引号或者双引号，就表示字符；如果数字没有单引号或者双引号，就表示数值。

如果单个字符按照一定的顺序排列，就构成了字符串序列，如“2018 年 1 月 1 日”。字符串序列构成的数据属于字符串类型。其中，字符串的元素是一个从左到右的顺序序列，每个位置的元素在计算机存储时都被自动“贴上”一个序号，该序号是从“0”开始。例如，字符串“2018 年 1 月 1 日”是由 9 个字符构成，从最左侧“2”到最右侧“日”分别用 0、1、2、3、4、5、6、7、8 序号表示元素位置。这样，我们可以根据元素位置读取元素内容。如例 2-8 所示，最左侧“2”和最右侧“日”分别位于序列的 0 位置和 8 位置。我们把“2018 年 1 月 1 日”字符串放入变量 a，可以利用 index()方法查看字符串中某字符的序号。a.index('2')就是查看字符 2 的位置；a.index('日')就是查看字符“日”的位置。如果要查看字符串中某个序号对应的字符，利用 a[x]，其中 x 是字符串序号。要查看字符串序号 4 对应的字符，利用 a[4]即可查找到字符“年”。要截取子字符串，利用 a[x:y+1]，其中 x 是起始序号，y 是结束序号。需要注意的是，序号 4 并不是字符串的第 4 个元素，而是第 5 个元素，因为序号是从 0 开始计数的。

**【例 2-8】** 查找字符串“2018 年 1 月 1 日”中字符“2”和字符“日”在字符序列中的位置，并给出序号 4 对应的字符是什么？序号 5～8 对应的字符串是什么？

```
a="2018年1月1日"
a1=a.index('2')
a2=a.index('日')
print (a1,a2)
```

```
a3=a[4]
print (a3)
a4=a[5:9]
print (a4)
```

运行结果：

```
08
年

"1月 1日"
```

### 1. 字符串的运算

一个或一串字母、数字、符号等加上一对单引号或者双引号就构成了一个字符或字符串。为了方便理解，我们将字符看作是只有一个字符的字符串。字符串是可以合并或复制。合并使用加法符号，复制使用乘法符号。

**【例 2-9】** 单字符合并与复制。

```
a='b'
b='o'
c='y'
d=a+b+c
e=d* 3
print (a,b,c,d,e)
```

运行结果：

```
b o y boy boyboyboy
```

**【例 2-10】** 字符串合并与复制。

```
a="butter"
b='fly'
c=a+b
d=c* 3
print (a,b,c,d)
```

运行结果：

```
butter fly butterfly butterflybutterflybutterfly
```

### 2. 字符串的处理

字符串是 Python 程序设计中常用的一种数据类型。我们可以对字符串进行各种处理，例如，截取子字符串、获取字符的 ASCII 码等操作。

如例 2-11 所示，已知十二个月的字符串“一月二月三月四月五月六月七月八月九月十月十一月十二月”，根据月份将字符串分离成 12 个字符串。我们已知这个字符串有 26 个字符，共有 26 个位置，从 0 到 25。从左到右，我们把第 1 个和第 2 个字符合并，第 3 个和第 4 个字符合并，以此类推。需要注意的是字符串“十一月”和“十二月”由三个字符构成。

**【例 2-11】** 将字符串“一月二月三月四月五月六月七月八月九月十月十一月十二月”按照月份分离成子字符串，即“一月”“二月”……“十二月”。

```
a='一月二月三月四月五月六月七月八月九月十月十一月十二月'
a1=a[0]+a[1]
a2=a[2]+a[3]
a3=a[4]+a[5]
a4=a[6]+a[7]
a5=a[8]+a[9]
a6=a[10]+a[11]
a7=a[12]+a[13]
a8=a[14]+a[15]
a9=a[16]+a[17]
a10=a[18]+a[19]
a11=a[20]+a[21]+a[22]
a12=a[23]+a[24]+a[25]
print(a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12)
```

运行结果：

```
一月 二月 三月 四月 五月 六月 七月 八月 九月 十月 十一月 十二月
```

ASCII 编码是最常用的字符编码，这种编码方式使用十进制数值 0～127 表示计算机键盘的常见字符和特殊字符，例如，使用 65～90 表示字符“A～Z”；97～122 表示字符“a～z”；48～57 表示字符“0～9”。可以利用标准函数 chr() 和 ord() 进行字符与 ASCII 编码之间的转换。chr(x) 函数返回 x 对应的字符；而 ord(y) 函数是返回 y 对应的编码。

**【例 2-12】** 张明刚进入大学，一位学长给了他一张纸条，上面写着：“87/101/108/99/111/109/101/33”。学长说，如果你猜出来这串密码是什么意思，将会送你一本书。刘明看出这是一串字符编码后，利用了一个简单函数，很快就破解出来。你知道刘明破解密码的方法吗？

```
A=chr(87)+chr(101)+chr(108)+chr(99)+chr(111)+chr(109)+chr(101)+chr(33)
print(A)
```

运行结果：

```
Welcome!
```

【例 2-13】 张明的旅行箱密码是三位数字，他通过对名字的拼音字母进行编码，把得到的码值相加而生成密码。你能猜出他的密码是什么吗？

```
A=ord('z')+ord('h')+ord('a')+ord('n')+ord('g')+ord('m')+ord('i')+
  ord('n')+ord('g')
print(A)
```

运行结果：

```
963
```

### 3. 字符串的内置函数

Python 语言为字符串设置了很多内置函数。内置函数就是安装 Python 后，不需要导入第三方模块即可直接使用的标准函数。关于字符串的内置函数很多，对于初学者，我们需要掌握的内容如表 2-2 所示。使用方法都是对某个字符串后加上句点“.”，再使用某个内置函数。

表 2-2 字符串的常用内置函数

| 内置函数 | 功能 | 示例 | 示例结果 |
|---|---|---|---|
| String.lower() | 将字符串的字符全部小写 | a='BJTU computer'<br>b=a.lower()<br>print(b) | bjtu computer |
| String.upper() | 将字符串的字符全部大写 | a='BJTU computer'<br>b=a.upper()<br>print(b) | BJTU COMPUTER |
| String.islower() | 判断字符串是否全部小写。如果是，返回 True，否则返回 False | a1='BJTU computer'<br>a2='lly'<br>b1=a1.islower()<br>b2=a2.islower()<br>print(b1,b2) | False True |
| String.isupper() | 判断字符串是否全部大写。如果是，返回 True，否则返回 False | a1='BJTU computer'<br>a2='lly'<br>b1=a1.isupper()<br>b2=a2.isupper()<br>print(b1,b2) | False False |

续表

| 内置函数 | 功　能 | 示　例 | 示例结果 |
|---|---|---|---|
| String.isnumeric() | 判断字符串是否全部是数字。如果是，返回 True，否则返回 False | a1='BJTU 2018'<br>a2='54321'<br>b1=a1.isnumeric()<br>b2=a2.isnumeric()<br>print(b1,b2) | False True |
| String.isspace() | 判断字符串是否全部是空格。如果是，返回 True，否则返回 False | a='BJTU 2018'<br>b=a.isspace()<br>print(b) | False |
| String.replace(old,new) | 将字符串的原字符被新字符替代 | a='BJTU 2017！'<br>b='2018'<br>c=a.replace(a[5:9],b)<br>print(c) | BJTU 2018！ |
| String.count(substring) | 子字符串 substring 在字符串 String 中出现的次数 | a='BJTU BJTU'<br>b=a.count('BJTU')<br>print(b) | 2 |

## 2.1.3　布尔类型

布尔值在程序设计中是至关重要的。布尔值经常与控制数据流的关系运算符和逻辑运算符一起使用。布尔值有两个，即“真”或“假”。“真”或“假”在计算机内部用二进制数“0”或“1”表示。通过布尔值，我们便可以根据变量的真假值来做出判断，以此来控制程序的路径走向，需要注意的是，Python 3.X 版本采用了 True 和 False 来表示真假，布尔值的首字母是大写的。例 2-14 中的“＞”和“＝＝”都是关系运算符。需要注意的是，“＝＝”用于比较左右两端的数据是否相等，要与赋值符号“＝”区别。

**【例 2-14】**　比较两个数字、两个字符串是否相等。



```
a=123;
b=321
c=a>b
print (c)
a='lly'
b='LLY'
c=a==b
print (c)
```

运行结果：

```
False
False
```

通过调用 bool 方法可以检查变量值的存在与否。将数据作为参数传给 bool 方法，返回 True 或者 False。空值或者 None 都会被认为是 False，而其他情形则被认为是 True。例 2-15 中变量值为空字符串，则返回值是 False；非空字符串返回值是 True；c 的值是 None，因此返回值是 False；如果是数值 1，则返回值是 True。

**【例 2-15】** bool 方法使用示例。

```
a=bool('')
b=bool('hello')
c=None
d=bool(c)
e=bool(1)
print (a,b,d,e)
```

运行结果：

```
False True False True
```

## 2.1.4 列表类型

列表是包含了 0 或多个元素的有序序列。在 Python 语言中，列表元素用方括号“[ ]”括起来，不同的元素之间用逗号“,”隔开。列表的长度没有限制，并且可以灵活控制列表，即增加元素、修改元素、删除元素。一般，当生成列表元素时，列表序号会在计算机内自动生成。序号的起始位置一般按照我们输入的顺序，从左到右按照 0，1，2，…的递增顺序标记。实际上，Python 语言规定了计算机内部也可以按照从右到左的顺序标记，这时是按照－1，－2，－3，…的递减顺序标记序号，如图 2-1 所示。两个列表 list1 和 list2 包括 4 个元素，如果 list1 在计算机内部从 B 开始标记序号，list2 在计算机内部从 2 开始标记序号，此时序号是递增的，分别为 0、1、2、3；如果 list1 在计算机内部从 U 开始标记序号，list2 在计算机内部从 8 开始标记序号，此时序号是递减的，分别为－1、－2、－3、－4。




图 2-1 列表元素与列表序号

列表的元素可以是单一的数据类型，如列表中每个元素都是数值类型数据，或者每个元素都是字符串类型数据，如图 2-1 所示。列表的元素也可以是混合类型的数据，如图 2-2 所示。列表 list3 中包含 4 个元素，分别是字符串类型、整数类型、浮点数类型、列表类型。

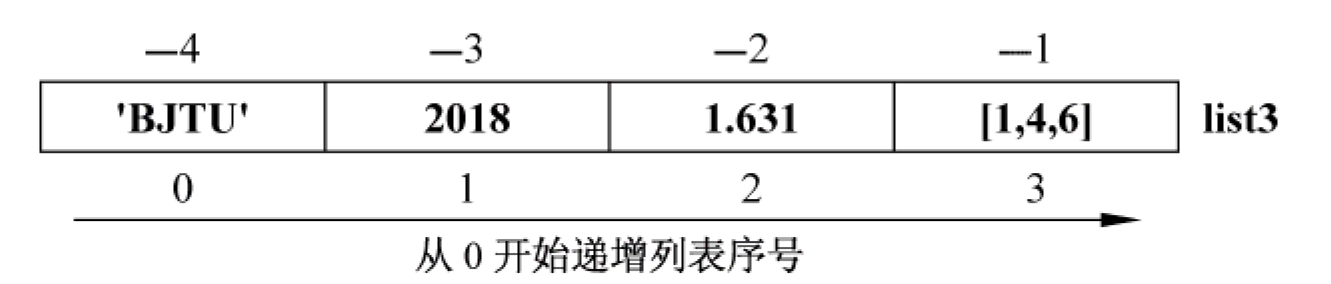



图 2-2　混合数据类型的列表

### 1. 不同类型的列表

单一类型列表，指的是列表中存储着数据类型相同数据的列表，这些数据可以是整型、字符串型、布尔型、浮点型等，如例 2-16 所示。混合类型列表与单一类型列表概念相对，指的是存储数据类型不同的数据的列表，如例 2-17 所示。在嵌入型列表中，又存储着数据类型为列表的数据，我们称为嵌套式列表，如例 2-18 所示。

**【例 2-16】**　单一类型列表使用示例。

```
list4= [1,2,3]
a1= list4[0]
a2= list4[1]
a3= list4[2]
a4= list4[- 1]
print (list4, a1,a2,a3,a4)
```

运行结果：

```
[1, 2, 3] 1 2 3 3
```

例 2-16 中列表的元素是包括数字 1、2、3，存储到列表 list4 中，计算机分配相应的存储空间。数值 1 存储在变量 list4[0]中，数值 2 存储在变量 list4[1]中，数值 3 存储在变量 list4[2]中。变量 list4[0]、list4[1]、list4[2]是三个有关联的变量，0、1、2 分别为列表序号。需要注意的是，列表序号是从 0 开始的。列表 list4 也可以从右至左标记序号，如 list4[－1]就是列表 list4 最右侧的元素，即数字 3。

**【例 2-17】**　混合类型列表使用示例。

```
list5= [1, 'Hello',2.13]
a1= list5[0]
a2= list5[1]
a3= list5[2]
a4= list5[- 2]
print (list5, a1,a2,a3,a4)
```

运行结果：

```
[1, 'Hello', 2.13] 1 Hello 2.13 Hello
```

例 2-17 中列表的元素包括 1、'Hello'、2.13，存储到 list5 中，计算机会根据不同的数据类型分配不同的存储空间。数字 1 存储在变量 list5[0]中，会分配 2 个字节；字符串'Hello'存储在变量 list5[1]中，会分配 5 个字节；小数 2.13 存储在变量 list5[2]中，会分配 4 个字节。变量 list5[0]、list5[1]、list5[2]是三个有关联的变量，0、1、2 分别为列表的序号。列表 list5 也可以从右至左标记序号，如 list5[－2]就是列表 list5 从右侧数第 2 个元素，即字符串'Hello'。

**【例 2-18】** 嵌套式列表使用示例。

```
list6= [1, 'Hello',[4,5,6]]
a1= list6[0]
a2= list6[1]
a3= list6[2]
a4= list6[2][0]
a5= list6[-1][-2]
a6= list6[2][-2]
print (list6, a1,a2,a3,a4,a5,a6)
```

运行结果：

```
[1, 'Hello', [4, 5, 6]] 1 Hello [4, 5, 6] 4 5 5
```

例 2-18 中列表的元素包括 1、'Hello'、[4,5,6]，存储到 list6 中，计算机会根据不同的数据类型分配不同的存储空间。其中，列表 [4,5,6]即是 list6 独立的一个元素，将[4,5,6]赋给变量 a3。我们利用 a3[0]、a3[1]、a3[2]的方式定位列表[4,5,6]中的每个元素，也可以利用 list6 定位子列表中的某个元素，如例题中使用了 list6[2][0]来定位元素 4。列表 list6 也可以从右至左标记序号，如 list6[－1][－2]就是列表 list6 从右侧数第 1 个元素，即列表[4,5,6]。[4,5,6]表示从右侧数第 2 个元素，即数字 5。列表嵌套中，两种序号的标记方法可以混合使用，例如 list6[2][－2]同样定位了数字 5，如图 2-3 所示。

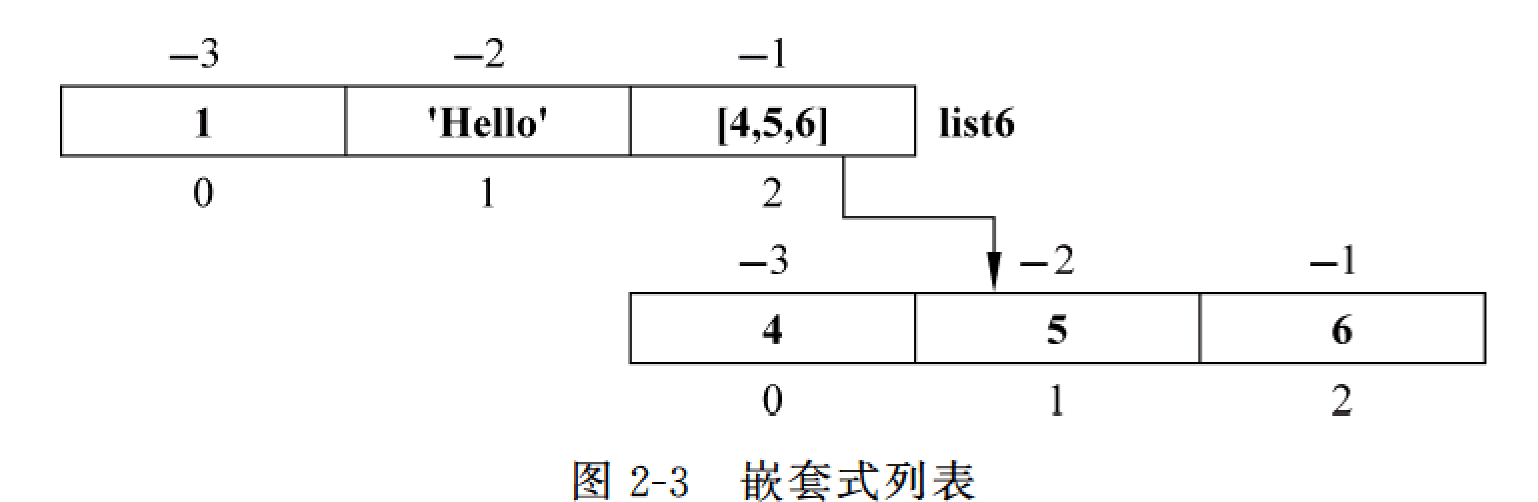



图 2-3　嵌套式列表

## 2. 列表常用操作

列表用方括号“[]”将要处理的数据括起来，每个元素按照顺序排列，列表根据序号调用每个元素。列表中元素可以被替换、被删除或者增加新元素。列表的常用操作如表 2-3 所示。

表 2-3 列表的常用操作

| 操作方法 | 功　能 | 示　例 | 示例结果 |
|---|---|---|---|
| list[i]=a | 列表 list 的元素被新数据替代。a 表示任意类型数据 | list7=[1,'Hello',[4,5,6]]<br>list7[0]='BJTU'<br>print(list7) | ['BJTU','Hello',[4,5,6]] |
| list[x:y+1]=list2 | 新列表替代原列表 list 从序号 x 到 y 的元素 | list8=[1,2,3,4,5]<br>list9=[2017,2018]<br>list8[2:4]=list9<br>print(list8) | [1,2,2017,2018,5] |
| list=list1 + list2 | 合并两个列表 list1 和 list2,构成新列表 list | list10=[1,2,3,4]<br>list11=[5,6]<br>list12=list10+list11<br>print(list12) | [1,2,3,4,5,6] |
| list=list1 * n | 复制 n 次列表 list1,构成新列表 list | list13=[1,2,3]<br>list14=list13 * 2<br>print(list14) | [1,2,3,1,2,3] |
| list=list. append(a) | 向列表 list 的末尾增加元素 a | list15=['B','J','T']<br>list15. append('U')<br>print(list15) | ['B','J','T','U'] |
| list=list. insert(i,a) | 向列表 list 序号为 i 的位置插入元素 a,原列表中序号 i 位置开始的元素后移 | list16=['B','T','U']<br>list16. insert(1,'J')<br>print(list16) | ['B','J','T','U'] |
| del list[x: y+1] | 删除列表 list 从序号 x 到 y 的元素 | list17=['B','J','T','U','X','Y']<br>del list17[4:6]<br>print(list17) | ['B','J','T','U'] |
| a=list. pop(i) | 删除原列表序号为 i 的元素,并将该元素取出 | list18=['B','J','T','U','X']<br>a=list18. pop(4)<br>print(list18,a) | ['B','J','T','U'] X |
| list1=list(a) | 将字符串 a 转化成列表 list1 | list19=list('BJTU')<br>print(list19) | ['B','J','T','U'] |

## 2.1.5 字典类型

列表是有序序列,要访问列表中的元素,可以通过列表序号。而现实生活中有很多的数据是“成对”出现的,例如,学生学号对应学生姓名;身份证号对应个人信息;通讯录中电话号码对应姓名。Python 语言中采用了“字典类型”保存和处理这些数据。字典类型利用了映射的思想将数据对建立关联,即设置了“键值”和“元素值”。要访问字典,不是通过

序号，而是通过“键值”。字典与序列不同，它可以无序存放。例如，通过键值“172301”可以访问序列的元素值“刘一”，“172302”访问元素值“张三”。字典由若干的键值对“键值：元素值”构成。Python 语言中通过大括号“{ }”建立字典，建立的方式如下。

```
{键值 1: 元素 1, 键值 2: 元素 2,…,键值 n: 元素 n}
```

键值和元素值通过冒号“：”连接，不同的键值用句号“，”隔开。其中，“键值”是唯一的不可重复的，但对应的元素值可以不唯一。在 Python 程序中，为同一键值多次定义了不同的元素值，按照字典中此键值出现的顺序，只取最右侧的一对“键值：元素值”，如例 2-19 所示。因此，需要注意的是字典的“键值：元素值”，一个键值只对应一个元素值信息。

**【例 2-19】** 为同一键值“010”赋予不同的元素值并查看字典。

```
dict1= {'010':'L','010':'L','010':'Y'}
dict2= {'010':'L','010':'Y','010':'L'}
print (dict1)
print (dict2)
```

运行结果：

```
{'010': 'Y'}
{'010': 'L'}
```

### 1. 字典增、改、删操作

如果向字典中添加一对“键值：元素值”，不能像列表那样利用“＋”对列表进行合并，以构成新列表。字典不支持“＋”合并，而是利用“dict[新键值]＝元素值”的方法添加到原字典中，如例 2-20 所示。

**【例 2-20】** 已知字典 dict＝{'010'：'刘一'，'011'：'张三'，'012'：'赵五'}，向字典中添加“'013'：'吴七'”。

```
dict= {'010':'刘一','011':'张三','012':'赵五'}
print(dict)
dict['013']= '吴七'
print(dict)
```

运行结果：

```
{'010': '刘一', '011': '张三', '012': '赵五'}
{'010': '刘一', '011': '张三', '012': '赵五', '013': '吴七'}
```

此外，还可以利用 update 的方法向字典添加新内容。其使用方法是将一个新的字典添加到原字典中，利用“dict1. update(dict2)”方法。我们可以将例 2-20 修改成例 2-21 所示。

**【例 2-21】** 已知字典 dict={'010': '刘一','011': '张三','012': '赵五'}，往字典中添加“'013': '吴七'”，利用 update()方法。

```
dict1={'010':'刘一','011':'张三','012':'赵五'}
print(dict1)
dict2={'013':'吴七'}
dict1.update(dict2 )
print(dict1)
```

运行结果：

```
{'010': '刘一', '011': '张三', '012': '赵五'}
{'010': '刘一', '011': '张三', '012': '赵五', '013': '吴七'}
```

如果要从字典中删除一对“键值:元素值”，可以利用类似列表的方法删除字典内容，利用“del dict[键值]”，如例 2-22 所示。如果遇到嵌套字典，即字典中“键值”对应的元素值又是字典类型，仍可以使用“del dict[键值][子键值]”方法，如例 2-23 所示。

**【例 2-22】** 已知字典 dict={'010': '刘一','011': '张三','012': '赵五'}，从字典中删除“'012': '赵五'”。

```
dict3={'010':'刘一','011':'张三','012':'赵五'}
print(dict3)
del dict3['012']
print(dict3)
```

运行结果：

```
{'010': '刘一', '011': '张三', '012': '赵五'}
{'010': '刘一', '011': '张三'}
```

**【例 2-23】** 已知字典 dict={1:{'010': '刘一','011': '张三','012': '赵五'},2:{'110' '王二','111': '李四','112': '孙六'}，从字典中删除“'012': '赵五'”。

```
dict4={1:{'010':'刘一','011':'张三','012':'赵五'},2:{'110':'王二','111':'李四','112':'孙六'}}
print(dict4)
del dict4[1]['012']
print(dict4)
```

运行结果：

```
{1: {'010': '刘一', '011': '张三', '012': '赵五'}, 2: {'110': '王二', '111': '李四', '112': '孙六
'}}
{1: {'010': '刘一', '011': '张三'}, 2: {'110': '王二', '111': '李四', '112': '孙六'}}
```

如果从字典中修改一对“键值:元素值”,可以利用类似字典添加的方法修改字典内容。修改过程中,只能改变键值对应的“元素值”,不能修改对应的键值。利用“dict[原键值]=新元素值”可以修改原字典元素值,如例 2-24 所示。

**【例 2-24】** 已知字典 dict={'010':'刘一','011':'张三','012':'赵五'},将字典中'赵五'修改成'李四'”。

```
dict3={'010':'刘一','011':'张三','012':'赵五'}
print(dict3)
dict3['012']='李四'
print(dict3)
```

运行结果:

```
{'010': '刘一', '011': '张三', '012': '赵五'}
{'010': '刘一', '011': '张三', '012': '李四'}
```

## 2. 字典常用操作

如果要将字典中的“键值”提取汇聚成列表,将字典中的“元素值”提取汇聚成列表,判断某对数据是不是在字典中,这些操作如何完成?我们可以利用一些内置函数完成字典的常用操作,如表 2-4 所示。

**表 2-4 字典的常用操作**

| 操作方法 | 功能 | 示例 | 示例结果 |
| --- | --- | --- | --- |
| dic.keys() | 提取字典 dic 的所有键值 | dict4={1:'北京',2:'上海',3:'广州'}<br>print(dict4.keys())<br>list_key=list(dict4.keys())<br>print(list_key) | dict_keys([1,2,3])[1,2,3] |
| dic.values() | 提取字典 dic 的所有元素值 | dict5={1:'北京',2:'上海',3:'广州'}<br>print(dict5.values())<br>list_value=list(dict5.values())<br>print(list_value) | dict_values(['北京','上海','广州'])['北京','上海','广州'] |

续表

| 操作方法 | 功　　能 | 示　　例 | 示例结果 |
|---|---|---|---|
| dic.items() | 返回所有对“键值:元素值” | dict6={1:'北京',2:'上海',3:'广州'}<br>print(dict6.items())<br>list_items=list(dict6.items())<br>print(list_items) | dict_items([(1,'北京'),(2,'上海'),(3,'广州')])<br>[(1,'北京'),(2,'上海'),(3,'广州')] |
| dic.get(k) | 根据键值 k 找到对应的元素值 | Dict7={1:'北京',2:'上海',3:'广州'}<br>a=dict7.get(1)<br>print(a) | 北京 |
| a=dic.pop(k) | 删除键值 k 对应的“键值:元素值”,并且提取此对“键值:元素值”给 a | dict8={1:'北京',2:'上海',3:'广州'}<br>a=dict8.pop(1)<br>print(dict8)<br>print(a) | {2: '上海',3: '广州'} 北京 |

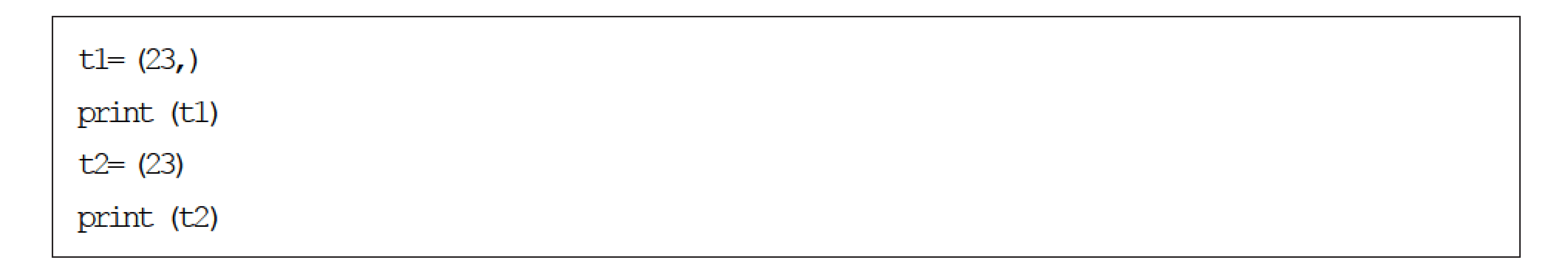
## 2.1.6　元组类型

元组是 Python 语言的一种数据类型,存储数据的方式与列表十分相似,但也有不同之处。元组使用小括号“( )”,内部元素可以是单一类型或混合类型,各个元素之间使用逗号“,”隔开。与列表最大的不同是元组具有不可变性,即不能更改元素值,元组初始化后就不能修改了。元组就像一个不可改变的列表。

### 1. 不同类型的元组

如果在一个元组中只存储了一个元素,这个元组即是单元素元组。需要在这个元素后面要加一个逗号,表示单元素元组,否则程序会认为是其他类型的数据。如例 2-25 所示,虽然结果类似,但 t1 是元组类型的数据,而 t2 只是整型类型的数据。

**【例 2-25】**　单元素元组使用示例。



```
t1= (23,)
print (t1)
t2= (23)
print (t2)
```

运行结果:

```
(23,)
23
```

单一类型元组只包含一种类型的元素,混合类型元组包含多种类型的元素,嵌入式元组中存储着数据类型为元组的数据,如例 2-26 所示。

【例 2-26】 单一类型元组、混合类型元组、嵌入式元组使用示例。

```
t1= (4,5,6,7)
t2= (7,8,9,'Smith')
t3= (4, (5,6,7))
print (t1,t2,t3)
a= t3[1][2]
print (a)
```

运行结果：

```
(4, 5, 6, 7) (7, 8, 9, 'Smith') (4, 5, 6, 7, 7, 8, 9, 'Smith') (4,(5,6,7))
```

## 2. 列表、字典、元组的综合训练

【例 2-27】 设计"学生课程信息管理系统"，原始数据如表 2-5 所示。完成下述问题：

（1）根据表 2-5 统计有多少名学生。

（2）如果要添加一名学生赵五的课程成绩，如何操作？

（3）计算学生每门课程的平均分和每名学生的课程平均分。

**表 2-5 原始数据**

| 学生学号 | 姓名 | 高数 | 英语 | 计算机 |
|---|---|---|---|---|
| 1801 | 张三 | 98 | 74 | 90 |
| 1802 | 李四 | 67 | 94 | 85 |
| 1803 | 孙六 | 71 | 86 | 83 |

（1）根据表 2-5 统计有多少名学生：

```
print('****************学生课程信息管理系统****************')
dict9= {'1801':('张三',98,74,90),'1802':('李四',67,94,85),'1803':('孙六',71,86,83)}
print(dict9)
```

运行结果：

```
****************学生课程信息管理系统****************
{'1801': ('张三', 98, 74, 90), '1802': ('李四', 67, 94, 85), '1803': ('孙六', 71, 86, 83)}
```

（2）添加一名学生赵五的课程成绩：

```
print('****************学生课程信息管理系统****************')
dict9= {'1801':('张三',98,74,90),'1802':('李四',67,94,85),'1803':('孙六',71,86,83)}
```

```
a= len(dict9)
print('共有',a,'名学生信息！')
```

运行结果：

```
****************学生课程信息管理系统****************
共有 3名学生信息!
```

（3）计算学生每门课程的平均分和每名学生的课程平均分：

```
print('****************学生课程信息管理系统****************')
dict9= {'1801':['张三',98,74,90],'1802':['李四',67,94,85],'1803':['孙六',71,86,83]}
sum_score1= dict9.get('1801')[1]+ dict9.get('1802')[1]+ dict9.get('1803')[1]
sum_score2= dict9.get('1801')[2]+ dict9.get('1802')[2]+ dict9.get('1803')[2]
sum_score3= dict9.get('1801')[3]+ dict9.get('1802')[3]+ dict9.get('1803')[3]
ave_score1= int(sum_score1/3)
ave_score2= int(sum_score2/3)
ave_score3= int(sum_score3/3)
print('高数平均分：  ',ave_score1)
print('英语平均分：  ',ave_score2)
print('计算机平均分:',ave_score3)
```

运行结果：

```
****************学生课程信息管理系统****************
高数平均分：  78
英语平均分：  84
计算机平均分：86
```

# 2.2　访问不同类型的数据

## 2.2.1　Python 语言常用符号

### 1. 单引号

Python 语言中用单引号括起来表示字符串，如例 2-28 所示。

**【例 2-28】** 单引号使用示例。

```
str= 'this is string'
print (str)
```

运行结果：

```
this is string
```

### 2. 双引号

Python 语言中用双引号括起来也可以表示字符串，与单引号的字符串用法完全相同。但当字符串中出现单引号时，要选择用双引号，如字符串“I'm a student”，如例 2-29 所示。

**【例 2-29】** 双引号使用示例。

```
str=" I'm a student"
print (str)
```

运行结果：

```
I'm a student
```

### 3. 三引号

利用三引号(''')，表示多行的字符串，可以在三引号中自由地使用单引号和双引号，如例 2-30 所示。

**【例 2-30】** 三引号使用示例。

```
str='''this is string.
this is 'pythod' string!
this is “string"! '''
print(str);
```

运行结果：

```
this is string
this is 'python' string
this is "string"!
```

### 4. 方括号

方括号([ ])除了列表类型使用，还可以通过序号定位列表、元组等数据类型中的元素，例如，list[3]表示定位列表 list 中序号为 3 的元素。此外，还可以在列表、元组中用于分片，例如，list[2:4]表示定位列表 list 中序号 2～3 的元素，如例 2-31 所示。

【例 2-31】 方括号具体使用示例。

```
t= (1,2,3)
L= [4,5,6]
print (t[1],L[2])
```

运行结果:

```
2 6
```

## 5. 括号

Python 语言中多处用到了小括号。定义元组时,可以利用小括号将元组各元素括起来。在 Python 程序设计中,算术表达式与数学四则运算法则一致,用小括号表示优先级。在 Python 3.X 中,print 命令变为 print()函数。在标准函数或自定义函数中,将变量或参数写入小括号内。关于函数的相关内容后续会详解。在 Python 语言中,大括号是定义字典类型数据时使用的符号,如例 2-32 所示。

【例 2-32】 括号使用示例。

```
t= (2018,'BJTU! ')
dict= {1:'中国',2:'美国',3:'英国'}
a= (3+ 4) * 6
b=bin(a)
print(t)
print(dict)
print(a)
print(b)
```

运行结果:

```
(2018, 'BJTU! ')
{1: '中国', 2: '美国', 3: '英国'}
42
0b101010
```

## 6. 冒号

Python 语言中多处用到了冒号。Python 语言定义函数时,出现在函数定义语句末尾。Python 语言在使用 if 语句、for 语句、while 语句等控制语句时,冒号出现在 if、for、while 语句末尾,表示满足某条件将执行后续语句,后续语句都应当缩进。此外,冒号可以出现在字典类型数据中,连接“键值:元素值”,如例 2-33 所示。

【例 2-33】 冒号使用示例。

```
def NewYear(a):
    print('今年是:',a)
a=2018
NewYear(a)

dict={1:'中国', 2:'美国', 3:'英国'}
for i in dict.keys():
    print(dict.get(i))
```

运行结果：

```
今年是:2018
中国
美国
英国
```

### 7. 逻辑运算符号

Python 语言中规定按位或“|”表示二进制中两个数，至少有一个为 1 时，结果都为 1，否则为 0。按位与“&”表示二进制中两个数，都为 1 时，结果为 1，否则为 0。按位异或“^”表示二进位中两个数，不相同时，结果为 1，否则为 0。按位取反“～”表示 x 的翻转是 －(x＋1)，即正数变负数，负数变正数，如例 2-34 所示。

【例 2-34】 逻辑符号使用示例。

```
a=23
b=99
c=a | b
d=a & b
e=a ^ b
f=~ a
print (c,d,e,f)
```

运行结果：

```
119   3   116   -24
```

### 8. 自变运算符号

Python 语言中规定“＋＝”表示先进行加法运算，再进行赋值。“－＝”表示先进行减法运算，再进行赋值。“＊＝”表示先进行乘法运算，再进行赋值。“/＝”表示先进行除法

运算，再进行赋值，如例 2-35 所示。

【例 2-35】 自变运算符号使用示例。

```
a=90
a+=45
b=43
b-=23
c=11
c *=7
d=63
d /=9
print (a,b,c,d)
```

运行结果：

```
135 20 77 7.0
```

## 9. 转义符

Python 语言中，用反斜杠“\”表示转义字符，如表 2-6 所示。

表 2-6 常用的转义符

| 转义符 | 功 能 | 示 例 | 示例结果 |
|---|---|---|---|
| \ | (在行尾时)续行符 | string='str_one'\ <br>'item_two'\ <br>'tem_three' <br>print(string) | str_oneitem_tw<br>otem_three |
| \\ | 反斜杠符号 | string='c:\\now' | c:\now |
| \' | 打印单引号符号 | string='I\'am<br>student.'<br>print(string) | I'am student. |
| \" | 打印双引号符号 | string="我想说:\"2018 年到了！\""<br>print(string) | 我想说:"2018 年到了!" |
| \n | 换行符号 | string="轻轻的我走了,\n 正如我轻轻的来;"<br>print(string) | 轻轻的我走了,正如我轻轻的来; |
| \t | 横向制表符 | string="高数\t 英语\t 计算机\t"<br>print(string) | 高数 英语 计算机 |

## 2.2.2 序列的操作

### 1. 索引操作

Python 语言中，字符串类型、列表类型、元组类型的数据都构成有序序列，此时的数据称为元素。这些元素按照输入的顺序从左至右，序号从 0 开始，程序设计中通常把这种处理序列的方式叫"索引"操作。"序号"有时也称为"索引号"。索引号从 0 开始，按照 0、1、2、3…顺序存储元素值，称为"正向索引"；索引号从右至左，即从 -1 开始，称为"反向索引"。如例 2-36 所示，从最后一个字符 m 开始。在索引操作中可以通过一些内置函数完成有序序列操作。例如，利用有序序列通用的内置函数 len()验证序列长度，如例 2-37 所示。

**【例 2-36】** 反向索引使用示例。

```
s='Spam'
print (s[-1],s[-3])
```

运行结果：

```
m p
```

**【例 2-37】** 索引使用示例。

```
s='Spam'
print (len(s),s[0],s[2])
```

运行结果：

```
4 S a
```

### 2. 分片操作

Python 语言中的序列除了简单地利用位置进行索引，也支持分片(slice)操作，这是一种一步就能够提取整个分片的方法。在一个分片中，左边界默认为 0，右边界默认为分片序列的长度，且负偏移量也可以用作分片的边界。S[a:b]指的是下标从 a 到 b 对应的元素，但不包括下标 b 对应的元素；S[a:]省略后面的数字，指的是从下标为 a 对应的元素到后面所有的元素；S[:b]指的是从字符串开头的元素直到下标 b 对应的元素，但不包括下标 b 对应的元素；S[:]指的是 S 从头到尾的元素。以上的操作并不改变字符串本身的内容，如例 2-38 所示。

【例 2-38】 分片使用示例。

```
S='spam'
print (S[1:3],S[1:],S[0:3])
print (S[:3],S[:-1],S[ : ])
```

运行结果：

```
pa pam spa
spa spa spam
```

作为一个序列，字符串也支持使用加号进行合并（将两个字符串合成为一个新的字符串），或者重复（使用乘法重复字符串的内容），如例 2-39 所示。

【例 2-39】 分片的合并和复制使用示例。

```
S='spam'
X=S+'xyz'
Y=S*8
print (S,X,Y)
```

运行结果：

```
spam spamxyz spamspamspamspamspamspamspamspam
```

### 3. 不可变性

Python 中元组类型的数据具有不可变性，即创建后值不能改变。比如，一般不能通过对某一位置进行赋值而改变元素值，如例 2-40 所示。Python 中的元组是一种类似于列表的类型，但列表是可变的，而元组不可变。元组本身是不可变的，但它所包含元素的可变性也取决于该元素的属性，如例 2-41 所示。

【例 2-40】 元组不可变性使用示例。

```
t=(23,33,49)
t[0]=32
print (t[0])
```

运行结果：

```
Traceback (most recent call last):
  File "C:\Users\RH\Desktop\buer.py", line 2, in<module>
t[0]=32
TypeError: 'tuple' object does not support item assignment
```

```
S='spam'
S='z'+S[1:]
S[0]='z'
print (S)
```

运行结果：

```
Traceback (most recent call last):
  File "C:\Users\RH\Desktop\buer.py", line 3, in<module>
S[0]='z'
TypeError: 'str' object does not support item assignment
```

**【例 2-41】** 元组与列表的相对不可变性使用示例。

```
t=(23,33,[43,53,63])
t[2][1]=153
print(t)
```

运行结果：

```
(23, 33, [43, 153, 63])
```

## 2.2.3 指定函数对序列的操作

对于字符串、元组、列表、字典等数据类型，Python 提供了很多函数用于访问、处理这些对象，表 2-7 列出了一些常用函数。

**表 2-7 Python 语言常用函数和模块**

| 功能模块 | | 功能描述 | 示例 | 示例结果 |
|---|---|---|---|---|
| 常用函数 | find() | 检测字符串中是否包含子字符串 str，如果包含，则返回位置序号，不包含，则返回－1 | S='spam'<br>a=S.find('pa')<br>print (a) | 1 |
| | replace() | 把字符串中的 old(旧字符串)替换成 new(新字符串) | S='spam'<br>a=S.replace('pa','xyz')<br>print (a) | sxyzm |
| | split() | 通过指定分隔符对字符串进行切片 | S='aa,bb,cc,dd'<br>a=S.split(',')<br>print (a) | ['aa',<br>'bb',<br>'cc',<br>'dd'] |

续表

| 功能模块 | | 功能描述 | 示例 | 示例结果 |
|---|---|---|---|---|
| 常用函数 | upper() | 将字符串中的小写字母转为大写字母 | S='spam'<br>a=S. upper()<br>print (a) | SPAM |
| | len() | 返回对象(字符、列表、元组等)长度或项目个数 | S='spam'<br>a=len(S)<br>print (a) | 4 |
| | append() | 在列表末尾添加新的对象 | L=['s','p','a','m']<br>L. append('xyz')<br>print (L) | ['s','p','a','m','xyz'] |
| | insert() | 将指定对象插入列表的指定位置 | L=['s','p','a','m']<br>L. insert(1,'xyz')<br>print (L) | ['s','xyz','p','a','m'] |
| | pop() | 移除列表中的一个元素(默认最后一个元素),并返回该元素的值 | L=['s','p','a','m']<br>L. pop()<br>print (L) | ['s','p','a'] |
| | remove() | 移除列表中某个值的第一个匹配项 | L=['s','p','a','m']<br>L. remove('p')<br>print (L) | ['s','a','m'] |
| | clear() | 删除字典或列表的所有元素 | L=['s','p','a','m']<br>L. clear()<br>print (L) | [ ] |
| | tuple() | 以一个序列为参数,将其转化为元组 | a=tuple('boring')<br>b=tuple(list('boring'))<br>print (a)<br>print (b) | ('b','o','r','i','n','g')<br>('b','o','r','i','n','g') |
| 常用第三方模块 | random 模块 | 可作为随机数字的生成器和随机选择 | import random<br>a=random. random()<br>b=random. choice([1,2,3,4])<br>print (a,b) | 0. 5976200600135281<br>3 |
| | math 模块 | math 模块实现了许多的数学运算函数 | import math<br>a=math. pi<br>b=math. sqrt(64)<br>print (a,b) | 3. 141592653589793<br>8. 0 |

### 1. 字符串的常用函数

(1) find(obj)检测字符串中是否包含子字符串 str。

(2) replace(obj1,obj2)实现将旧字符串替换成新字符串。

(3) split(symbol)通过指定分隔符对字符串进行切片。

需要注意的是，由于字符串的不变性，这些函数操作都不是改变原始的字符串，而是创建一个新的字符串作为结果。

### 2. 列表的常用函数

列表有很多方法可以使用，下面简单介绍几种常见的方法。

(1) list.append(obj)在列表末尾添加新的对象。

(2) list.insert(index,obj)将对象插入列表。

(3) list.pop(obj=list[-1])移除列表中的一个元素(默认最后一个元素)，并返回该元素的值。

(4) list.remove(obj)移除列表中某个值的第一个匹配项。

### 3. 字典的常用函数

(1) dict.pop(key)的参数是字典中的已有键，移除该键对应的值并返回该值，如果字典中没有该参数，则出现 KeyError 错误。

(2) clear()用于删除字典内所有元素。

(3) len(dict)计算字典元素个数，即键的总数。

### 4. 引用第三方的数学模块

除了表达式，随 Python 分发的还有一些常用数学模块。例如，math 模块，以及可作为随机数字的生成器和随机选择的 random 模块等。

## 2.2.4 字典遍历

字典遍历可以依次访问字典中的每个数据。与元组、列表两种类型最大的不同就是，字典中的数据是以“键值:元素值(key-value)”形式存在的，所以字典遍历有多种方法。

### 1. 遍历字典的键值 key

遍历字典中的每个“键值”方式如例 2-42 所示。

**【例 2-42】** 遍历字典的键值示例。

```
d= {'Mary':56,'Bart':90,'Paul':89}
for key in d:
    print (key)
```

运行结果：

```
Mary
Bart
Paul
```

### 2. 遍历字典的元素值 value

遍历字典中的每个“元素值”方式如例 2-43 所示。

**【例 2-43】** 遍历字典的元素值示例。

```
d={'Mary':56,'Bart':90,'Paul':89}
for value in d.values():
    print (value)
```

运行结果：

```
56
90
89
```

### 3. 遍历字典的项

遍历字典中的每一项，即遍历每对“键值:元素值”，如例 2-44 所示。

**【例 2-44】** 遍历字典的每项示例。

```
d={'Mary':56,'Vart':90,'Paul':89}
for item in d.items():
    print (item)
```

运行结果：

```
('Mary', 56)
('Vart', 90)
('Paul', 89)
```

### 4. 遍历字典的 key-value

同时并且分别遍历字典中的“键值”和“元素值”，如例 2-45 所示。

**【例 2-45】** 遍历字典的“键值”和“元素值”示例。

```
d={'Mary':56,'Vart':90,'Paul':89}
for key,value in d.items():
    print(key,value)
```

运行结果：

```
Mary 56
Vart 90
Paul 89
```

需要注意的是，如果想判定字典中是否存在某对“键值:元素值”，可以使用 in、not in 操作符，如例 2-46 所示。

**【例 2-46】** 遍历字典的 in 和 not in 操作符示例。

```
dict={'Mary':56,'Vart':90,'Paul':89}
if 'Mary' in dict:
    print ('键 Mary存在')
if 'Lily' not in dict:
    print ('键 Lily不存在')
```

运行结果：

```
键 Mary存在
键 Lily不存在
```

# 2.3 表达式与运算符

什么是操作符？以表达式“8+6/2”等于 11 为例，这里 8、6、2 被称为操作数，+号和/号被称为操作符。但在程序设计中，经常把“操作符”称为“运算符”。Python 语言支持的运算符主要有以下几种类型：算术运算符、比较(即关系)运算符、逻辑运算符、位运算符、赋值运算符。

## 2.3.1 算术符号与算术表达式

算术表达式通常是由常量、变量、函数、圆括号、算术运算符等组成。表达式的运算过程与数学运算中的规则一样，有括号先运算括号内的子表达式。有多层括号，先运算最里层。同一层，负号优先运算，其次运算乘除，最后是加减；同一优先级从左到右进行运算。算术符号与算术表达式如表 2-8 所示。

**表 2-8 算术符号与算术表达式**

| 算术运算符 | 功 能 | 示 例 |
|---|---|---|
| + | 加法运算 | 4+5 返回 9 |
| - | 减法运算 | 7-4 返回 3 |
| * | 乘法运算 | 6*9 返回 54 |
| % | 取余运算 | 8%3 返回 2 |
| // | 取整除，返回商的整数部分 | 9//2 返回 4 |
| ** | 返回 x 的 y 次幂 | 2**3 返回 8 |

### 2.3.2 关系符号与关系表达式

关系表达式用于比较算术表达式的值大小或者两个数的大小，参与比较的符号称为关系符号。常用的关系符号有两类，一类表示相等与不等关系的符号，即等号、不等号等；另一类表示大小关系的符号，即大于号、小于号、不大于号、不小于号等。关系符号与关系表达式如表 2-9 所示。

**表 2-9 关系符号与关系表达式**

| 算术运算符 | 功　　能 | 示　　例 |
| --- | --- | --- |
| ＝＝ | 等于，比较对象是否相等 | 5＝＝5 返回 True |
| !＝ | 不等于，比较两个对象是否不相等 | 7!＝4 返回 True |
| ＞ | 大于，返回 x 是否大于 y | 6＞9 返回 False |
| ＜ | 小于，返回 x 是否小于 y | 9＜3 返回 False |
| ＞＝ | 大于等于，返回 x 是否大于或等于 y | 8＞＝3 返回 True |
| ＜＝ | 小于等于，返回 x 是否小于或等于 y | 9＜＝2 返回 False |

### 2.3.3 逻辑符号与逻辑表达式

逻辑表达式是用逻辑运算符将关系表达式或逻辑量连接起来的有意义的式子。逻辑表达式的值是一个逻辑值，即“True”或“False”。Python 语言的编译系统在给出逻辑运算结果时，以字符串 True 表示“真”，以字符串 False 表示“假”。Python 采用“and”“or”“not”表示逻辑与、逻辑或和逻辑非。逻辑符号两端的操作对象称为操作数。逻辑符号与逻辑表达式如表 2-10 所示。

**表 2-10 逻辑符号与逻辑表达式**

| 逻辑运算符 | 功　　能 | 示　　例 |
| --- | --- | --- |
| and | 如果两个操作数都是真的，那么返回 True，否则返回 False | (5＞3) and (3＜2) 返回 False<br>(5＞3) and (3＞2) 返回 True |
| or | 只要左侧操作数 True，无论右侧操作数为 True 还是 False，即返回 True；若左侧为 False，右侧 True，则返回 True，否则返回 False | (5＞3) or(3＜2) 返回 True<br>(5＜3) or(3＜2) 返回 False<br>(5＜3) or(3＞2) 返回 True |
| not | 用于反转操作数的逻辑状态，如果一个条件为 True，则逻辑非运算符将返回 False | not (5＞3) 返回 False<br>not(3＜2) 返回 True |

### 2.3.4 位运算符与位运算

按位运算是把表达式中的数字转换为二进制数字进行运算的一种形式。在计算机系

统中，数值一律用补码来表示。例如，2 的补码是 0010，3 的补码是 0011，计算“0010&0011”时，把 0010 和 0011 从右至左按位对齐进行二进制的与运算，结果为 0010，十进制数为 2。位运算符如表 2-11 所示。

**表 2-11　位运算符**

| 逻辑运算符 | 功　　能 | 示　　例 |
|---|---|---|
| & | 按位与运算符：参与运算的两个值，如果两个相应位都为 1，则该位的结果为 1，否则为 0 | 2 & 3 返回 2 |
| \| | 按位或运算符：只要对应的二个二进位有一个为 1 时，结果位就为 1 | 2\|3 返回 3 |
| ^ | 按位异或运算符：当两对应的二进位相异时，结果为 1 | 2^3 返回 1 |
| ～ | 按位取反运算符：对数据的每个二进制位取反，即把 1 变为 0，把 0 变为 1 | ～61 返回－62 |
| ＜＜ | 左移动运算符：运算数的各二进位全部左移若干位，由“＜＜”右边的数指定移动的位数，高位丢弃，低位补 0 | 3＜＜2 返回 12 |
| ＞＞ | 右移动运算符：把“＞＞”左边的运算数的各二进位全部右移若干位，“＞＞”右边的数指定移动的位数 | 12＞＞2 返回 3 |

## 2.3.5　运算符的优先级

运算符的优先级顺序按从高到低排列如表 2-12 所示。

**表 2-12　运算符优先级**

| 优先级别 | 运　算　符 | 描　　述 |
|---|---|---|
| 高 | ** | 幂(提高到指数) |
| ↓ | ～、＋、－ | 按位翻转，一元加号和减号 |
| | *、/、%、// | 乘，除，取模和地板除 |
| | ＋、－ | 加法和减法 |
| | ＞＞、＜＜ | 左，右按位转移 |
| | & | 位'AND' |
| | ^、\| | 按位异“或”和定期“或” |
| | ＜＝、＜、＞、＞＝ | 比较运算符 |
| | ＜、＞、＝＝、!＝ | 等式运算符 |
| | ＝、%＝、/＝、//＝、－＝、＋＝、*＝、**＝ | 赋值运算符 |
| 低 | not、or、and | 逻辑运算符 |

## 2.4 变量赋值与输出

### 2.4.1 直接赋值

与 C 语言不同,Python 语言一个很大的特点是,不需要声明变量的类型。C 语言是事先为变量定义好一个数据类型,根据这个数据类型的特点分配存储空间。Python 是给变量赋给"什么数据类型"的值,那么变量就是什么数据类型的。例如,如果定义了 a=5,那么 a 就是数值类型中的整型;如果定义了 a=4.5,那么 a 就是数值类型中的浮点型;如果定义了 a='Beijing',那么 a 就是数值类型中的字符串类型,如表 2-13 所示。

给变量赋值,除了用这种赋值语句的方式,还可以从磁盘中导入数据,给变量赋值。

**表 2-13 赋值运算符**

| 赋值运算符 | 功　　能 | 示　　例 |
|---|---|---|
| = | 简单的赋值运算符：右边的值赋给左边的变量 | a=4+5<br>print(a)返回 9 |
| += | 加法赋值运算符：右边的值加上左边变量的值再赋给变量 | a=2&3<br>a =34<br>print(a)返回 36 |
| *= | 乘法赋值运算符：右边的值乘以左边变量的值再赋给变量 | a=9-4<br>a * =4<br>print(a)返回 20 |
| /= | 除法赋值运算符：左边的值除以右边变量的值再赋给变量 | a=7!=4<br>a/=9<br>print(a)返回 0.1111111111111111 |
| -= | 减法赋值运算符：左边的值减去右边变量的值再赋给变量 | a=5 * 9<br>a-=34<br>print(a)返回 11 |
| %= | 取模赋值运算符：左边的值取余右边变量的值再赋给变量 | a=20+3<br>a%=11<br>print(a)返回 1 |
| **= | 幂赋值运算符：将左边变量的右边值的次方赋给变量 | a=4<br>a**=2 * 2<br>print(a)返回 256 |
| //= | 取整除赋值运算符：左边的值取整右边变量的值再赋给变量 | a=56<br>a//=34<br>print(a)返回 1 |

## 2.4.2 input()输入方式

Python 提供了一个 input()函数，可以让用户输入字符串和数值，并存放到一个变量中。运行时，Python 会出现一个交互式命令行等待你的输入。这时，用户可以输入任意字符，然后按回车键后完成输入。输入完成后，不会有任何提示。有一个问题，用户刚刚输入的内容到哪去了？答案是存放到 name 变量里了。可以使用 print(变量名)查看变量内容，如例 2-47 所示。

**【例 2-47】** input()输入示例。

```
a= input()
print (a)
name1= input()
print (name1)
name2= input()
print (name2)
从键盘输入：
10
运行结果：
10
从键盘输入：
'Paul'
运行结果：
'Paul'
从键盘输入：
Paul
运行结果：
Paul
```

需要注意的是，给变量赋值，可以通过表达式赋值，也可以通过调用函数赋值，还可以通过从键盘输入的方式给变量赋值。

## 2.4.3 eval()函数

eval()的功能是“将任意含有数字的字符串转化为有效的数字，进而参与表达式求值运算或者直接返回数字结果”。需要注意的是，待转化的字符串只能包含数字字符，不应该含有其他字符。eval()的使用方法如表 2-14 所示。

表 2-14 eval()的常用方法

| 常用方法 | 示 例 | 示例结果 |
|---|---|---|
| 仅包含字符串转换成数字 | string="1980"<br>print(string)<br>#查看类型,"str"表示字符串类型<br>print(type(string))<br>year=eval(string)<br>print(year)<br>#查看类型,"int"表示整数类型<br>print(type(year)) | 1980<br><class 'str'><br>1980<br><class 'int'> |
| 仅包含列表字符串转换成列表 | string="[1,2,3,4,5,6,7,8,9,10]"<br>list1=eval(string)<br>#提取列表 list1 的第 4 个元素<br>number=list1[3]<br>print(list1,number)<br>#查看类型,"list"表示列表类型<br>print(type(list1)) | [1,2,3,4,5,6,7,8,9,10] 4<br><class 'list'> |
| 仅包含字典字符串转换成字典 | string="{1:'赵',2:'钱',3:'孙'}"<br>dict1=eval(string)<br>print(dict1)<br>#查看类型,"dict"表示字典类型<br>print(type(dict1)) | {1: '赵',2: '钱',3: '孙'}<br><class 'dict'> |
| 仅包含元组字符串转换成元组 | string="(1,2,3,4,5,6,7,8,9,10)"<br>tuple1=eval(string)<br>print(tuple1)<br>#查看类型,"tuple"表示列表类型<br>print(type(tuple1)) | (1,2,3,4,5,6,7,8,9,10)<br><class 'tuple'> |
| 包含数学表达式字符串的运算 | number1=eval ("5"+"1 * 2")<br>#字符串拼接后为"51 * 2"<br>number2=eval ("5 * 1"+"2")<br>#字符串拼接后为"5 * 12"<br>print(number1,number2) | 102 60 |

## 2.4.4 format()输出方式

Python 提供了一个输出格式化字符串的函数,即 str.format()函数,str 是预输出的字符串。format()函数可以通过"{ }"和":"来输出各种格式的字符串。"{ }"指定了字符串中使用数据的序号,按照序号用对应的数据替代。例如,format()函数的替代顺序与数据对应的方式如下所示,得到的结果为"2018 年,选修了 5 门课程!",其中第一个"{ }"用 format()函数的第一个数据"'2018'"替代,第二个"{ }"用 format()函数的第二个数据"5"替代。

```
"{}年,我选修了{}门课程!".format("2018",5)
```

“{ }”的数据内容可以用很多种方法替代,str. format()的数据常用替代方法如表 2-15 所示。

**表 2-15 format()的输出数据替代方法**

| 替 代 方 法 | 示　　例 | 示 例 结 果 |
|---|---|---|
| 通过位置替代 | print('{0},{1}'. format('carmen',20))<br>print('{},{}'. format('carmen',20))<br>print('{1},{0},{1}'. format('carmen',20)) | carmen,20<br>carmen,20<br>20,carmen,20 |
| 通过变量替代 | print('{name},{age}'. format(age = 20, name = 'carmen')) | carmen,20 |
| 通过列表序号替代 | list1=['carmen',20,'China']<br>print('Hello, my name is {0[0]} from {0[2]}, and my age is {0[1]}. '. format(list1)) | Hello, my name is carmen from China, and my age is 20. |
| 通过字典键值替代 | dict1={'name':'carmen','age':20,'country':'China'}<br>print('Hello, my name is {name} from {country}, and my age is {age}. '. format(**dict1)) | Hello, my name is carmen from China, and my age is 20. |

“:”指定了字符串输出的样式,控制样式的命令放在“:”的右侧,“:”的左侧则是替代的具体数据。例如,format()函数控制输出样式如下所示,得到的结果为“今年,我的平均分为 96. 32!”,其中“. 2f”表示输出数据的小数部分精度为 2 位小数。

```
"今年,我的平均分为{: .2f}!".format(96.3213)
```

“:”左侧可以有很多种控制字符串样式的方式,format()的样式常用控制方法如表 2-16 所示。

**表 2-16 format()的输出常用样式**

| 常 用 样 式 | 示　　例 | 示 例 结 果 |
|---|---|---|
| 控制数据的小数部分精度 | print("pi 的值为{:. 4f}". format(3. 1415926)) | pi 的值为 3. 1416 |
| 控制数据的整数千位分隔 | print("今年的盈利额 $ {:,}". format(31415926)) | 今年的盈利额 $ 31,415,926 |
| 右侧对齐 | print("{:>15}年实现了预计目标!". format(2017)) | 2017 年实现了预计目标! |
| 居中使用“ * ”填充空格 | print("{: * ^15},我来了!". format(2018)) | *****2018***** * ,我来了! |
| 输出十进制、二进制、八进制、十六进制数据 | print('{:d}'. format(10)) # 十进制 10<br>print('{:b}'. format(10)) # 二进制 1010<br>print('{:o}'. format(10)) # 八进制 12<br>print('{:x}'. format(10)) # 十六进制 a | 10<br>1010<br>12<br>a |

# 本章小结

(1) 了解 Python 常用的数据类型。
(2) 熟练掌握序列的分片和索引操作。
(3) 熟练掌握各类常用函数对字符串、元组、列表、字典等数据类型的操作。
(4) 熟练运用 3 种方法遍历字典。
(5) 掌握常用的运算符,算术运算符、关系运算符、位运算符、逻辑运算符。
(6) 掌握 Python 的 input()、eval()等标准函数。

# 习　　题

## 1. 简答题

(1) 简述变量与常量的区别,并说明使用变量的必要性。
(2) 简述数据结构、数据类型和抽象数据类型的区别与联系。
(3) 简述元组、列表、字典的不同和相同点。
(4) 解释 Python 运算符/与//、* 和**的区别。

## 2. 填空题

(1) 在 Python 中,常用的数据类型有数值类型、________、________、________、________、________和________等。
(2) 使用 math 模块前,需要使用________语句导入该模块。
(3) 数学表达式$\frac{\lambda^k}{k!}e^{-\lambda}$的 Python 表达式为________。
(4) "[2] in [1,2,3]"返回的结果是________。
(5) 命题"x 小于或等于 y,且大于 z"的 Python 表达式是________。
(6) 命题"x 小于或等于 y,或大于 z"的 Python 表达式是________。
(7) 命题"x 是 y 的倍数"的 Python 表达式是________。
(8) 输入__name__返回的结果是________。
(9) "BBJJTTUU"[:2]返回的结果是________。
(10) 若字典 d={1:"a",2:"b"},则 sum(d)返回的结果是________,sum(d.keys())返回的结果是________,sum(d.values())返回的结果是________。

## 3. 选择题

(1) 下列选项中不合法的标识符是【　　】。
A) -　　B) class　　C) a&b　　D) 3x

(2) 若 a=math.e**(1j * math.pi),则 1+a 输出的结果是【　　】。

A) 0　　B) 1　　C) 2　　D) 以上都不是

(3) 函数 type(pow(-1,0.5)-1j+0xf * 2.718)返回的结果是【　　】。

A) <class 'complex'>　　B) <class 'int'>

C) <class 'long'>　　D) <class 'float'>

(4) len("BJTU")返回的结果是 4,len("北京交大")和 len("北京交大\nBJTU")返回的结果分别是【　　】。

A) 4,9　　B) 4,8　　C) 8,13　　D) 8,14

(5) 若字符串 s="BeijingJiaoTongUniversity",与 s[0:-1]不仅输出结果相同而且具有相同含义的是【　　】。

A) s[:-1]　　B) s[:]　　C) s[:len(s)-1]　　D) s[:len(s)]

(6) 设元组 t=(2,3),则 t[1]=1 返回的结果是【　　】。

A) (2,1)　　B) (1,3)　　C) TypeError　　D) NameError

(7) 设列表 l=[3],则 l * 3 返回的结果是【　　】。

A) [9]　　B) TypeError　　C) [3],[3],[3]　　D) [3,3,3]

(8) 以下会返回错误的语句是【　　】。

A) d1={}　　B) d2={0:1}

C) d3=dict([0,1],[2,3])　　D) d4=dict(([0,1],[2,3]))

(9) 表达式 25//3-100//5**2%3 * 5 的值为【　　】。

A) 0　　B) 8　　C) 3　　D) 1

(10) 下列表达式中与数学表达式$\frac{ab}{cd}$不对应的是【　　】。

A) a * b/c/d　　B) a * b/(c * d)　　C) a/c * b/d　　D) a * b/c * d

(11) 下列表达式非法的是【　　】。

A) 71//7　　B) 71.7//1.7　　C) 1+7j/3j　　D) 3j/j

(12) 表达式 type(range(9))返回的结果是【　　】。

A) <class 'range'>　　B) <class 'list'>

C) <class 'tuple'>　　D) <class 'str'>

(13) 若 list1=["name","address","postcode"],list2=["BJTU","Haidian","100044"],下列能使上述列表转换为元组的语句是【　　】。

A) tuple("list1")　　B) dict((list1))

C) tuple(list2)　　D) dict(key=list1,value=list2)

(14) 若字典 d={1:"a",2:"b"},则 len(d)返回的结果是【　　】。

A) 10　　B) 2　　C) 6　　D) 4

(15) 若字典 d={1:"a",2:"b"},则能够访问 d 的第一个元素的语句是【　　】。

A) d[0]　　B) d[1]　　C) d["0"]　　D) d["1"]

(16) 若集合 s={1,1,1,2,2,2,3,3,3},则 sum(s)和 len(s)的返回值是【　　】。

A) 18,9　　B) 6,3　　C) 18,3　　D) 6,9

# 第 3 章 文件操作

第 1 章和第 2 章编写的 Python 程序，其运行结果会显示在计算机屏幕上，但关闭计算机后就会消失。如果要再次查看程序的运行结果，必须再次运行程序。如果能够将程序运行后的数据保存下来，就可以随时查看或使用这些数据。计算机一般将运行后的数据保存到文件中。

## 3.1 认识文件

文件是存储在硬盘、CD、DVD、闪盘或其他存储介质上的数据集合。文件可以保存的数据形式多种多样，如图片、文本、字符、视频、音乐、动画、可执行程序等。了解文件，需要知道它的三个属性，即文件名、文件类型、文件位置。这三个属性分别说明文件叫什么、是什么、放在哪。可以想象自己的衣柜：不同衣服有各自的名称，就像文件名；它们属于不同的类型，如衬衣、外套、裤子、裙子等，就像文件类型；它们悬挂或摆放在衣柜中的不同格子里，这些格子就是衣服的位置，就像文件位置。

### 3.1.1 文件名

保存文件或者从文件中读取数据时，需要指定文件名称。文件名的形式如下：

```
*****.扩展名
```

文件名包含一个句点“.”，句点右侧部分指出了文件的类型，称为扩展名（extension name）。例如“.txt”表示一个文本文件，“.mp3”表示一种音乐文件，“.exe”表示可执行程序文件。Python 程序文件的扩展名为“.py”。句点左侧是用户根据 Python 命名规则和个人习惯为文件起的名字。

需要注意的是，在文件名或文件夹名中，最多可以有 255 个字符，其中包含驱动器和完整路径信息，因此用户实际使用的字符数小于 255。文件名或文件夹名中不能出现的字符包括“\、/、:、*、?、#、”、<、>、|”。而且文件名不区分英文字母大小写，例如“A1”与“a1”是同一个文件名。另外，文件名和文件夹名中可以使用汉字，例如“北交大.doc”。

### 3.1.2 文件类型

要对文件进行操作，必须明确文件的类型。文件包括文本文件和二进制文件类型。例如“. txt”文件和扩展名为“. py”的 Python 文件，都是纯文本文件，只包含基本的文本字符，如字母、数字、标点符号和一些特殊字符，以及换行符，但不包含字体、大小和颜色信息。它们可以用 Windows 的记事本打开。除此之外的所有其他文件类型都称为二进制文件，例如字处理文档(Word 文档)、PDF、图像、音乐、电子表格、可执行程序等。不同类型的二进制文件，都各自有专门的处理方式和工具，如图片处理软件、音乐播放软件等。

文本文件和二进制文件的区别是文本文件由一个个字节(Byte)组成，标准 ASCII 码使用一个字节中的七位来表示信息，可表示 128 种字符。而二进制文件使用的是连续二进制位(bit)，即二进制文件是按二进制的编码方式来存放文件的。文本文件和二进制文件的区别如图 3-1 所示。

数字：12345

| 文本文件存储 | 二进制文件存储 |
| --- | --- |
| “1”：00110001 | 110000 0011 |
| “2”：00110010 | |
| “3”：00110100 | |
| “4”：001101011 | |
| “5”：001101011 | |

图 3-1　文本文件和二进制文件的区别

### 3.1.3 文件位置

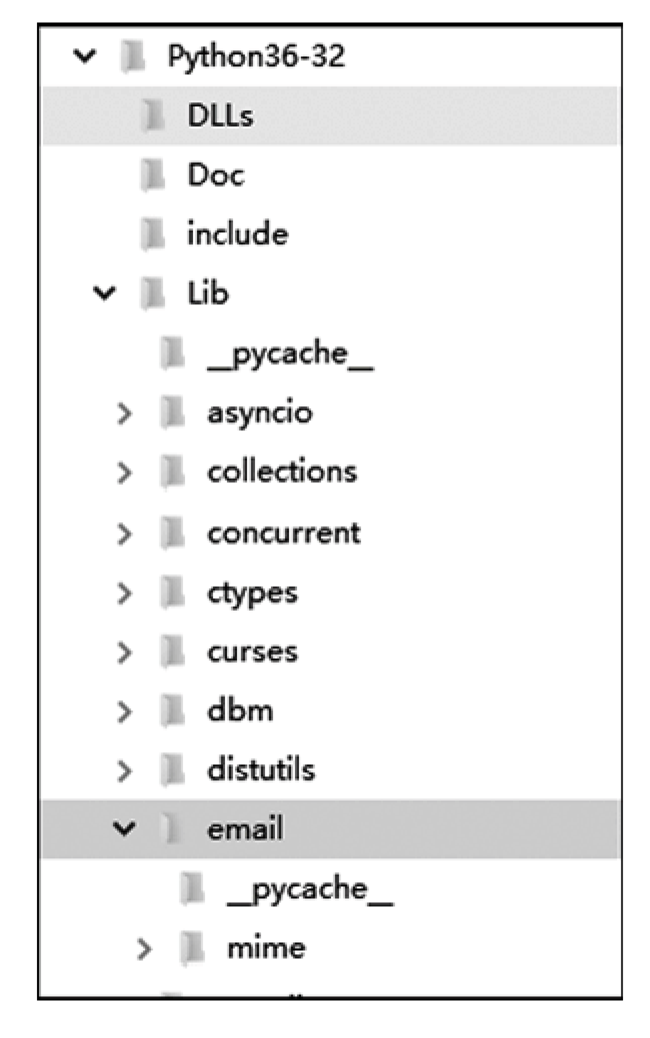
每个文件都要保存在存储介质中的某个地方，存储介质通常是硬盘。在 Windows 系统中，用文件夹和目录来组织文件夹结构或目录结构。目录结构就像一棵树，驱动器本身是树根，每个主文件夹是树干，每个主文件夹中的子文件夹就像小树杈，一层一层分解下去，直到最后的树叶，即为文件。

图 3-2 展示了一个目录结构。如果想要表示文件 generator. py 的位置，可以采用如下方式：

```
C:\Python36-32\Lib\email\generator.py
```

这种形式称为路径，它描述了文件在目录结构中的位置。该路径的意思是：从根目录 C 盘开始，进入名称为 Python36-32 的文件夹，在此文件夹下进入名称为 Lib 的子文件夹，在该子文件夹下进入下一层名为 email 的子文件夹，最后在这个子文件夹中包含了一个名称为 generator. py 的文件。其中的反斜杠“\”是路径分隔符，用于区分不同的文件





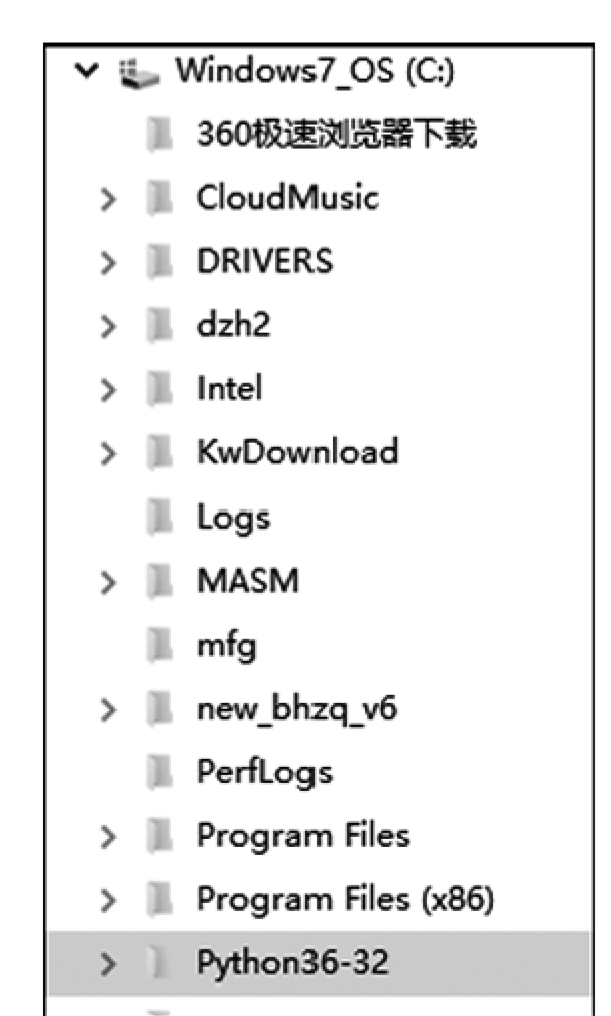



图 3-2　目录结构

夹。Windows 系统在路径名中可以使用斜杠"/"和反斜杠"\"两种分隔符，但在 Python 中，由于反斜杠"\"表示转义符，因此在表示文件路径时，通常使用双反斜杠"\\"以避免歧义，如下所示。

```
C:\\Python36-32\\Lib\\email\\generator.py
```

如果路径包括从根目录开始这个路径上的所有文件夹名，称为"绝对路径"。使用绝对路径可以保证找到文件，但有时该路径太长，引用不便。因此可以采用"相对路径"来表示文件位置。相对路径指的是程序所在的当前工作目录(cwd)。大多数操作系统都有"当前工作目录"，这是文件夹结构中目前所在的目录。如图 3-2 所示，如果当前目录为 C:\Python36-32\Lib\email，要找到文件 header.py，因为它就在当前目录下，可以采用相对路径，也就是直接使用文件名"header.py"。

# 3.2　文件的操作

在了解了文件名和文件路径后，就可以指定文件的位置，进行文件的读写操作。在 Python 中，二进制文件和文本文件的操作步骤是相同的：首先打开文件，获得该文件的使用权，此时文件处于占用状态，其他程序(进程)不能访问这个文件；然后对文件进行读写操作；操作完成后关闭文件，释放对文件的控制权，这样才能让其他程序(进程)访问这个文件。

## 3.2.1　文件的打开与关闭

Python 用 open()函数打开文件。例如，打开一个保存在 C:\Python32-36 下的名称

为 my_file.txt 的文本文件：

```
My_file=open('C:\\Python32-36\\my_file.txt', 'r')
```

其中，My_file 是一个变量，用来保存 open 函数返回的文件对象，之后所有的文件读写操作都使用这个文件对象完成。open 函数有两个参数，第一个参数指定要打开的文件，可以使用相对路径或绝对路径。第二个参数“r”表示打开文件的模式。open 函数有 6 种基本打开模式，如表 3-1 所示。

**表 3-1　Python 文件打开模式**

| 序号 | 打开模式 | 含　义 |
|---|---|---|
| ① | 'r' | 读模式，打开文件的默认方式。如果文件不存在，返回异常 |
| ② | 'w' | 写模式，如果文件不存在则创建新文件，存在则完全覆盖原文件 |
| ③ | 'a' | 追加写模式，如果文件不存在则创建，存在则在原文件末尾追加内容 |
| ④ | 'b' | 以二进制文件模式打开文件(可以添加到其他模式中) |
| ⑤ | 't' | 以文本文件模式打开文件。一般默认方式是文本文件模式 |
| ⑥ | '＋' | 与 r/w/a 一同使用，在原功能基础上增加读或写功能 |

文件打开模式用字符串表示，单引号或双引号均可，其中“＋”参数可以与其他模式联合使用，表明允许同时读和写。当打开二进制文件时，需要用“rb”参数，打开文本文件可以用“rt”或者不用参数。

文件操作结束后一般使用 close()函数关闭，以释放文件的控制权。使用方法如下：

```
<文件对象>.close()
```

例如，关闭 my_file.txt 文件的操作为“My_file.close()”。

## 3.2.2　读取文件

读取文件有 3 种方法：如果希望读取整个文件的内容，可以使用 read()方法；要逐行读取文件使用 readline()方法；要把读取内容保存为列表，使用 readlines()方法。例如，文本文件 my_file.txt 保存在 C:\Python32-36 目录下，内容如图 3-3 所示。

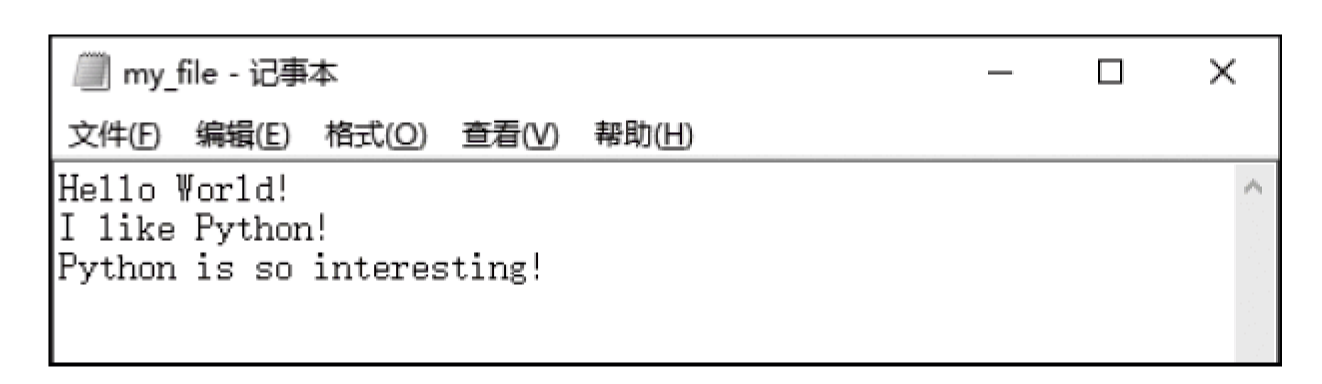



图 3-3　my_file.txt 文件

要将全部文件内容读取出来，如例 3-1 所示。

**【例 3-1】** 读取文件 my_file.txt 的内容。

```
file='my_file.txt'                    #定义文件路径(相对路径)
My_file=open(file,'r')                #打开文件,返回文件对象
strVar=My_file.read()                 #读取文件全部内容
My_file.close()                       #关闭文件
print(strVar)                         #显示字符串 strVar 内容
```

运行结果：

```
Hello World!
I like Python!
Python is so interesting!
```

由于该文件与 Python 程序在同一文件夹下，因此可以使用相对路径。也可以用 readline()方法从文件中读取一行，如例 3-2 所示。

**【例 3-2】** 读取一行 my_file.txt 的内容。

```
file='my_file.txt'                    #定义文件路径(相对路径)
My_file=open(file,'r')                #打开文件,返回文件对象
line=My_file.readline()               #读取文件一行内容
My_file.close()                       #关闭文件
line()
```

运行结果：

```
'Hello World!\n'
```

打开文本文件时，文件指针位于文件第一行的开始处。readline()读取文件第一行“Hello World!\n”，并保存为字符串 line，同时指针移动到该行的结尾处。字符串中的“\n”是换行符，表示一行的结束。从键盘输入字符串，在一行结束时按下回车键，就会在字符串结尾添加一个换行符。readline()会将这个换行符作为单独的字符读取出来。如果重复使用 readline()，则可以读取文件的每行内容，当所有行读取完毕时 readline()会返回一个空字符串。因此，readline()方法可以结合循环结构逐行读取文件内容，例如下面的程序：

```
file='my_file.txt'
My_file=open(file,'r')
line=My_file.readline()
while line!=" ":                #没有读取完毕,继续读取下一行
```

```
    print(line, end="")
    line=My_file.readline()
My_file.close()
```

运行结果：

```
Hello World!
I like Python!
Python is so interesting!
```

在多次使用 readline()后，如果希望文件指针回到文件的起始位置，可以使用 seek()方法：

```
seek(offset,whence)
```

seek()方法将文件指针移动到 offset 指定的位置。offset＝0，则移动到文件起始位置；offset＝1 代表指针当前所在位置；offset＝2，则移动到文件结尾。可选参数 whence 表示从哪个位置开始移动文件指针，默认值是 0，表示默认从文件起始位置移动，1 表示从当前位置移动，2 表示从文件结尾处移动。

**【例 3-3】** 利用 seek()指定 my_file. txt 某行的内容。

```
first_line=My_file.readline()
second_line=My_file.readline()
My_file.seek(0)          #文件指针移动到文件起始位置
new_line=My_file.readline()
new_line
```

运行结果：

```
"Hello World!\n"
```

代码中 My_file. seek(0)将文件指针移动到文件第一行的起始位置，因此 new_line 读取的是文件第一行的内容。

如果想一次获得文件各行内容，可以使用 readlines()方法。例如执行下列代码(假设文件已经打开)：

```
lines=My_file.readlines()
lines
```

运行结果：

```
['Hello World!\n', 'I like Python!\n', 'Python is so interesting!'] "Hello World!\n"
```

从结果可以看出，readlines()方法也会读取整个文件，但返回的是一个字符串列表，列表中的每个字符串就是文本中的每一行。当文件内容读取到列表中后，就可以采用各种处理方法，如排序、分片访问、删除、打印、转换数据类型等。

## 3.2.3 写入文件

写入文件就是向文件中添加内容，可以用 write()方法。向文件中添加内容有两种方式：写和追加。

写文件或追加文件的流程与读文件类似，先要用 open()函数打开文件，但模式参数不同。读文件时使用“r”，写文件时需要使用“w”，追加文件时需要使用“a”。当使用“w”时，如果文件已经存在，则文件中原有内容会被新写入的内容覆盖。如果文件不存在，则会自动创建一个新文件，向其中写入内容。使用“a”参数，新内容会添加到原有内容的后面，但文件必须是已经存在的文件，否则会提示出错，如例 3-4 所示。

**【例 3-4】** 向文件 my_file.txt 追加一行“Do you like Python?”。

```
file='my_file.txt'                              #文件路径(相对路径)
My_file=open(file,'a')                          #以追加模式打开文件,返回文件对象
My_file.write('\n Do you like Python? ')        #添加新行
My_file.close()
```

代码运行后，打开 my_file.txt 文件可以看到，文件中新增加了一行内容，如图 3-4 所示。请注意，在上文的 readlines()示例中，最后一个字符串是没有换行符“\n”的。write()不像 print()那样会自动添加换行符，因此在追加新字符串时需要在字符串开头手工添加一个换行符，这样新添加的内容会另起一行，否则新字符串会添加到最后一行“Python is so interesting!”之后。另外，由于计算机内存的访问速度远远快于磁盘的访问速度，Python 分配了一块名为“缓冲区”(buffer)的内存空间，用来临时保存将要写进磁盘的数据。一旦缓冲区满了，或者文件被关闭，缓冲区里的内容就会被写入磁盘，因此执行 write 操作后，必须关闭文件，才能确保所有数据在物理层上传输到磁盘中了。

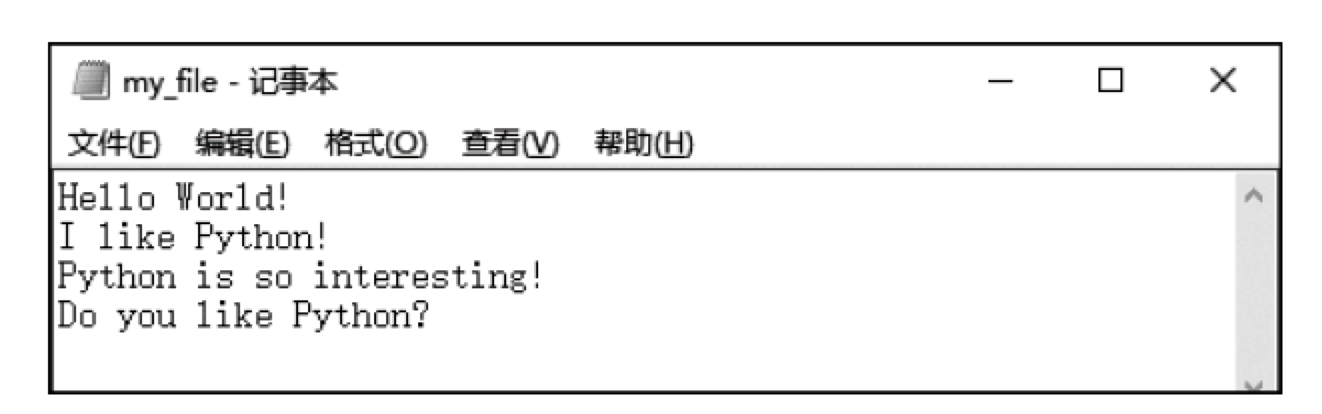



图 3-4 添加新内容的 my_file.txt 文件

如果用“w”模式打开 my_file.txt 文件，此时文件指针会指向文件第一行的开始处，新写入的内容会覆盖原来的内容。例如，如下代码写入字符串“Let's study Python together!”：

```
file='my_file.txt'                  #文件路径(相对路径)
My_file=open(file,'w')              #以写模式打开文件,返回文件对象
My_file.write('Let's study Python together!\n')         #写入新内容
My_file.close()
```

代码执行后，my_file. txt 文件如图 3-5 所示。

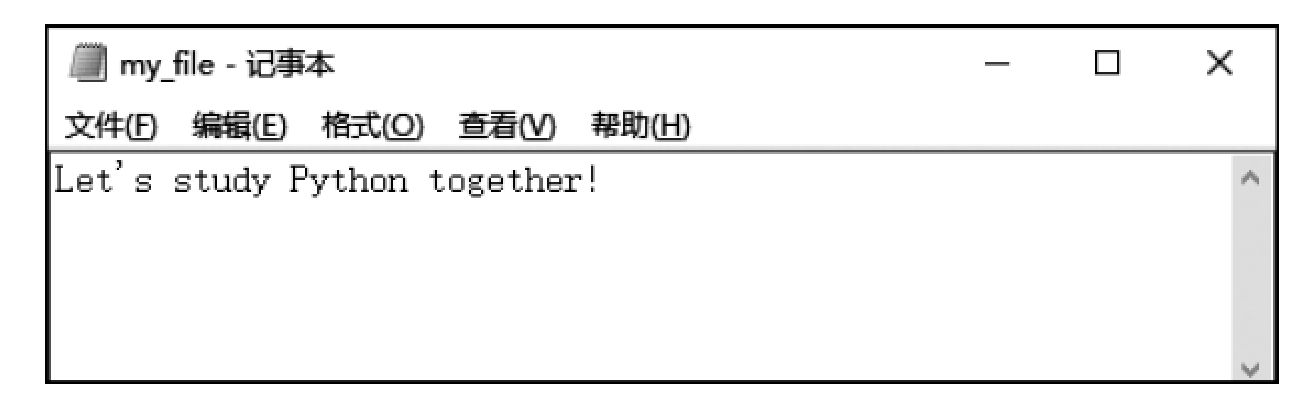



图 3-5　写入新内容的 my_file. txt 文件

还可以用 writelines()方法向文件写入一个列表。首先用 readlines()方法打开 my_file. txt 文件(文件内容如图 3-4 所示)，再将列表写入一个新文件 my_new_file. txt，并逐行显示文件内容，代码如下。

```
file='my_file.txt'
My_file=open(file,'r')
lines=My_file.readlines()                   #读取 my_file.txt 内容
My_file.close()
My_newfile=open('my_new_file.txt','w+')     #以读写模式创建新文件
My_newfile.writelines(lines)                #将 my_file.txt 内容写入新文件
My_newfile.seek(0)                          #文件指针移动到新文件起始位置
for line in My_newfile:
    print(line, end="")                     #逐行显示新文件内容
My_newfile.close()
```

代码运行后，会在当前目录下创建一个新文件 my_new_file. txt，其内容为列表 lines 中的内容，如图 3-6 所示。为了显示新文件的内容，在打开新文件时采用了“w+”参数，这样既能向新文件写入内容，又可以读取文件内容。执行 writelines()后，文件指针移动到文件结尾处，这时想要显示文件内容，会返回空字符串，因此使用 seek(0)将指针移动到文件开头，再逐行读取并显示文件内容。

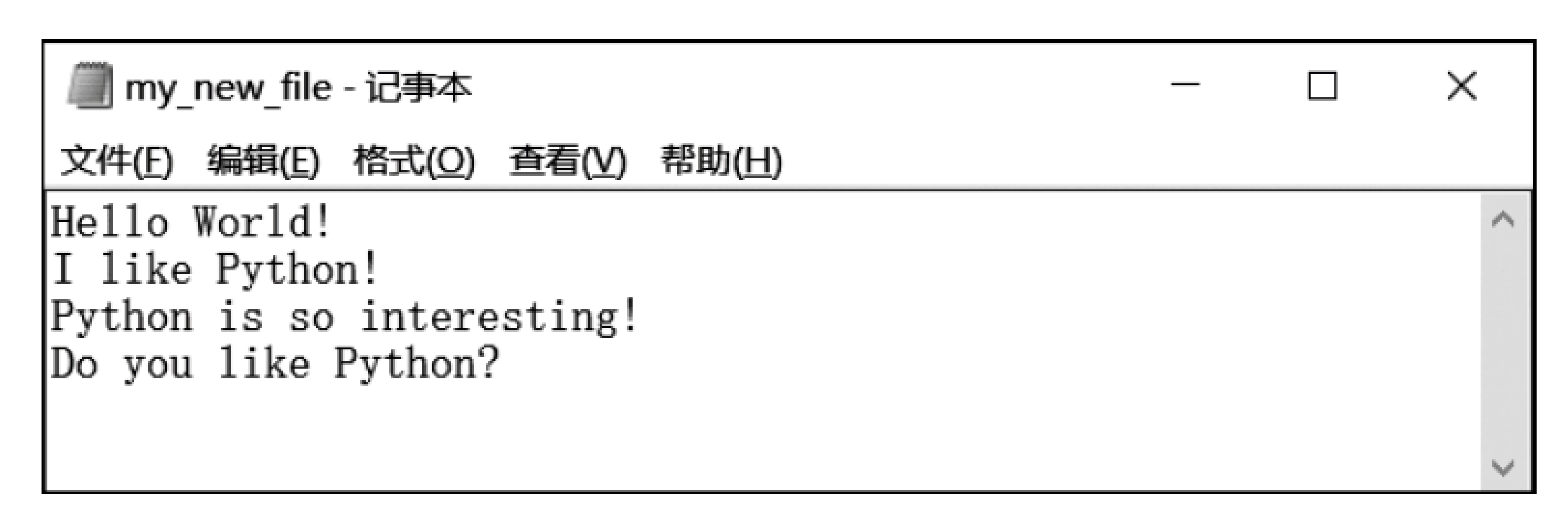



图 3-6　新创建的 my_new_file. txt 文件

# 3.3 图像文件和网络文件

3.2 节主要介绍了文本文件的读写方法，当处理图像、网页等二进制文件时，需要借助一些第三方库来解析它们。

## 3.3.1 图像文件的读写

### 1. PIL 库简介

PIL(Python Imaging Library)是 Python 中最常用的第三方图像处理库。PIL 库支持图像存储、显示和处理，它能够处理几乎所有图片格式，可以完成对图像的缩放、剪裁、叠加，以及向图像添加线条、图像和文字等操作。PIL 需要通过 pip 工具安装。在 DOS 命令模式下运行下列指令：

```
:\>pip install pillow
```

PIL 库包括 21 个类，其中 Image 类是 PIL 库中一个非常重要的类，任何一个图像文件都可以用 Image 对象表示。

### 2. PIL 的使用方法

首先要用 import 导入 Image 模块，然后通过 Image 类中的 open 方法载入图像文件。如果载入文件失败，会提示 IOError；如果载入文件成功，open()会返回一个 Image 对象。

图像文件的操作与文本文件类似，首先要打开文件，创建图像文件对象，然后进行处理。例 3-5 所示的程序显示了图像文件的以下三个常用属性。

(1) format：识别图像格式或来源，如果图像不是从文件中读取的，返回 None。

(2) size：图像的宽度和高度(单位为像素)，返回二元元组(宽度，高度)。

(3) mode：图像的颜色模式，L 代表灰度图像，RGB 代表真彩色图像，CMYK 代表印刷图像。

程序运行结果显示图像 ai.jpg 的格式是 jpg，来源于文件，宽度 640 像素，高度 347 像素，颜色模式是 RGB 真彩色。

**【例 3-5】** 打开图像文件 C:\Python32-36 文件夹下的 ai.jpg，获取这张图像的格式、图像所占的宽度和高度、图像的颜色模式。

```
#从 PIL 库导入 Image 模块
from PIL import Image
#打开 ai.jpg,返回 Image 对象
im= Image.open('c:\\Python32- 36\\ai.jpg')
```

```
#显示该图像的三个属性 JPEG (640, 347) RGB
print(im.format, im.size, im.mode)
```

运行结果：

```
JPEG (640,347) RGB
```

## 3.3.2　图像文件的处理

图像文件打开后，可以使用 Image 类中定义的各种方法进行操作。

### 1. 图像的显示

Image 类的 show()方法显示最新打开的图像，例如：

```
from PIL import Image
im=Image.open('c:\\Python32-36\\ai.jpg')
im.show()                #显示原图像
```

打开后的 aj.jpg 如图 3-7 所示。



图 3-7　显示图像 ai.jpg

### 2. 图像的保存

save()用于保存图像，有两个参数：文件名 filename 和图像格式 format。如果调用时不指定保存格式，将自动根据图像文件的扩展名保存图像；如果指定格式，则按照格式存储。save()可以实现图像格式的转换，例如：

```
from PIL import Image
im=Image.open('ai.jpg')
im.save('ai.png')
```

代码运行后会在当前目录下出现一个新的 ai.png 文件，相当于将.jpg 文件转换成了.png 文件。但是一般来讲，save()主要用于将临时的图像对象保存到硬盘上。转换格式可以使用功能更强大的 convert()方法。

### 3. 图像的复制与粘贴

crop(box)从图像中复制一个矩形图像。参数 box 是一个四元元组，四元素分别代表矩形左上角和右下角顶点的横纵坐标。坐标系统原点(0,0)为图片左上角。paste(region,box)将一个图像粘贴到另一个图像上。region 指被粘贴图像对象，变量 box 指定粘贴区域，如果是二元元组，代表粘贴区域左上角的横纵坐标；如果是四元元组，则代表左上和右下角的横纵坐标。如果为空，则默认为(0,0)。如果给定四元元组，被粘贴图像的尺寸必须与粘贴区域尺寸一样。如果尺寸不匹配，被粘贴的图像将被转换为当前图像的模式。代码如下：

```
from PIL import Image
im= Image.open('ai.jpg')
box= (500, 100, 640,347)              #定义复制区域
region= im.crop(box)                  #复制图像,返回新的图像对象
region.show()
im.paste(region, (0,0))               #将图像 region粘贴到图像 im的左上角
im.save('newai.jpg')                  #将剪裁下来的图像粘贴到原始图像上
im1= Image.open('newai.jpg')
im1.show()
```

代码运行后，原始图像 ai.jpg 中以坐标(500,100)和(640,347)为左上角和右下角的矩形区域(虚线框)被复制下来。再将复制的图像粘贴到原始图像上以原点(0,0)为左上角的区域，如图 3-8 所示。




图 3-8　图像的复制和粘贴

### 4. 图像的缩放与旋转

resize(size)根据参数 size 指定的尺寸调整图像，生成一个副本文件。rotate(angle)按照参数 angle 指定的角度逆时针旋转图像，生成一个副本文件。transpose()方法预定义了一些旋转方式，如左右反转、上下翻转、逆时针旋转(90°、180°、270°)等。

```
from PIL import Image
im= Image.open('ai.jpg')
im. resize((200,100))                          #缩小尺寸
out= im.rotate(45)                             #逆时针旋转 45°
out= im.transpose(Image.FLIP_LEFT_RIGHT)       #左右反转
out= im.transpose(Image.FLIP_TOP_BOTTOM)       #上下翻转
out= im.transpose(Image.ROTATE_180)            #逆时针旋转 180°
```

图像处理后的效果如图 3-9(a)～(d)所示。


(a) 缩小（200×100）


(b) 逆时针旋转45°


(c) 左右反转


(d) 逆时针旋转180°


(e) 上下翻转

图 3-9　图像的缩放和翻转

### 5. 图像的颜色变换

上文提到图像有不同颜色模式。在 RGB 模式下，每张图片由三个颜色通道 R、G、B 叠加而成。可以使用 split()分离三个颜色通道，对每种颜色分别处理，再用 merge()把几个通道合并形成新的图像。还可以使用 convert()将图像转换为不同的颜色模式。

```
im= Image.open('ai.jpg')
r,g,b= im.split()                      #分离三个颜色通道
im= Image.merge('RGB', (b,g,r))        #互换 b、r通道后合成新图像
im.show()
```

图像处理后的效果如图 3-10 所示。



图 3-10　图像的颜色变换

### 6. 图像的过滤与增强

PIL 中的 ImageFilter 模块和 ImageEnhance 模块提供了过滤图像和增强图像的方法。ImageFilter 模块预定义了 10 种图像过滤方法，可以提取图像轮廓、图像锐化、图像平滑等，主要使用 filter()方法来实现。ImageEnhance 模块专门用于图像的增强处理，可以增强(或减弱)图像的亮度、对比度、色度等。

```
from PIL import Image
from PIL import ImageFilter
from PIL import ImageEnhance
im= Image.open('ai.jpg')
detfilter= im.filter(ImageFilter.DETAIL)          #图像细节增强
confilter= im.filter(ImageFilter.CONTOUR)         #图像轮廓效果
smtfilter= im.filter(ImageFilter.SMOOTH)          #图像平滑
sharpfilter= im.filter(ImageFilter.SHARPEN)       #图像锐化
enhbrightim= ImageEnhance.Brightness(im)
brightness= 1.5
enh_bri= enhbrightim.enhance(brightness)          #图像亮度增强为原来的 1.5倍
enhcontrastim= ImageEnhance.Contrast(im)
contrast= 1.5
enh_con= enhcontrastim.enhance(contrast)          #图像对比度增强为原来的 1.5倍
```

图像处理效果如图 3-11 所示。

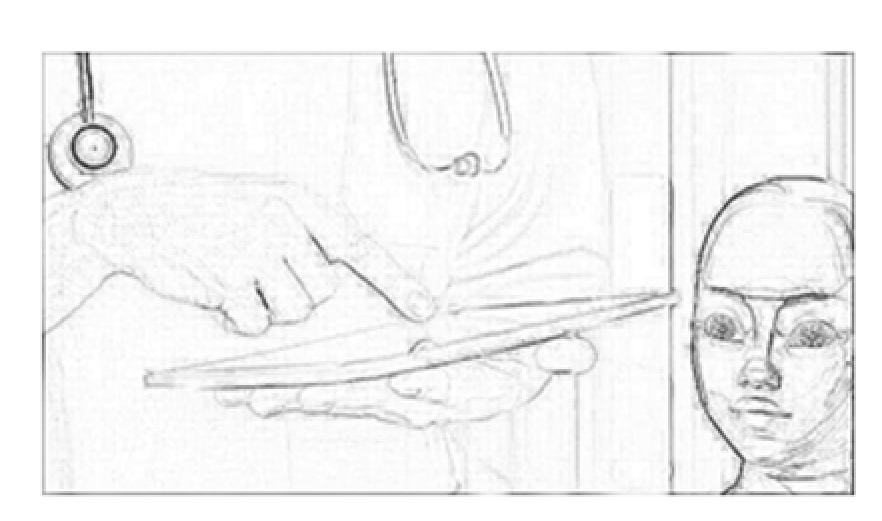
PIL 还支持对像素点的直接操作。例如将 ai.jpg 分离成三个颜色通道，对其中一个通道进行加强或减弱操作，再使用 Merge 将通道合并，从而改变图片的色调(冷暖色调的互换)等。如果要把 b 层每个像素点的亮度增大 20%，可以用下列代码：

```
im= Image.open('ai.jpg')
r,g,b= im.split()                        #分离三个颜色通道
out=b. point(lambda i:i * 1.2)           #lambda 表达式返回 i 的 1.2 倍
im= Image.merge('RGB',(r,g,b))           #重新合并通道后生成新图像
im.save()
```

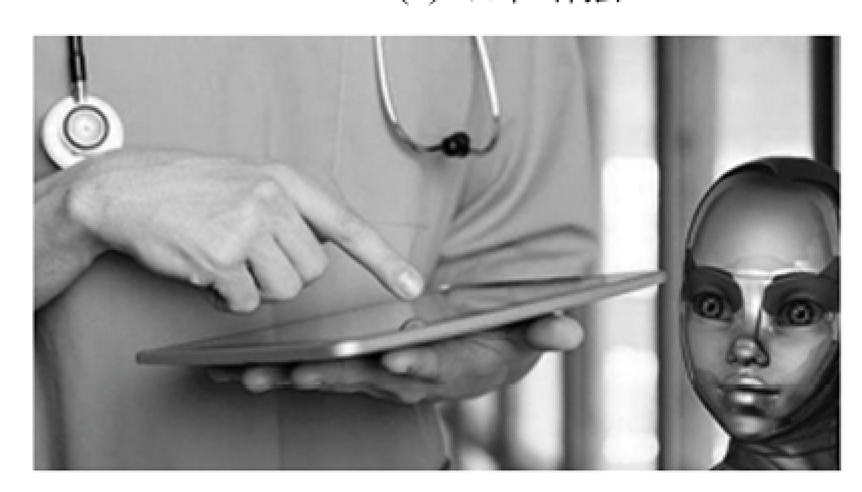
其中的 i 代表图片中的每个像素值，括号中的 lambda 表达式计算新的像素值，这里是原值的 1.2 倍。图像处理效果如图 3-12 所示。



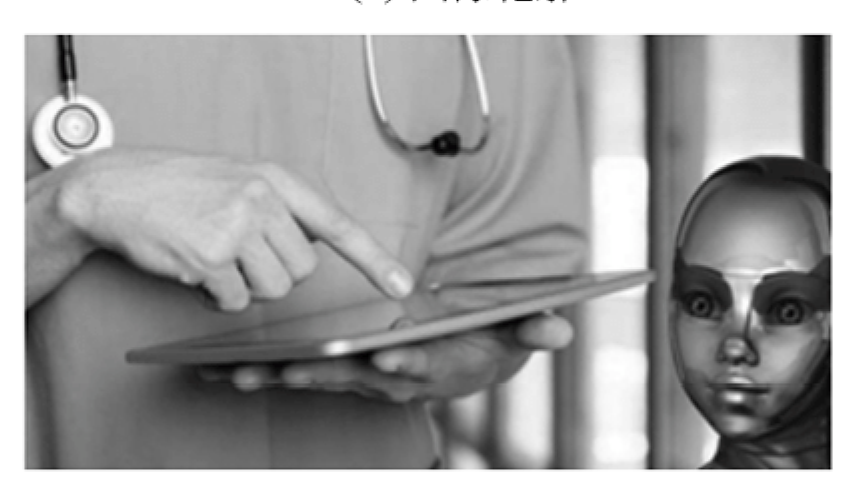
(a) 细节增强



(b) 图像轮廓



(c) 图像锐化



(d) 图像平滑



(e) 图像亮度增强1.5倍



(f) 图像对比度增强1.5倍

图 3-11　图像的过滤与增强



图 3-12　图像像素的处理

### 7. 序列图像的处理

序列图像是将多帧图像保存在一个图像文件中,按照一定时间间隔播放形成动画效果。我们常见到的 GIF 图片就是序列图像。PIL 对这种动态图片也提供了基本的处理方法。当用 open()打开这类图像文件时,会自动载入第一帧图像,使用 seek()和 tell()方法可以在各帧之间移动。seek(frame)表示跳转到指定的图像帧,tell()返回当前帧的序号。例如,读取 watson.gif 并将各帧图像保存为 PNG 文件,代码如下:

```
from PIL import Image
im= Image.open('watson.gif')                         #读入一个 GIF文件
try:
    im.save('frame{:02d}.png'.format(im.tell()))      #保存图像帧为 png文件
while True:
    im.seek(im.tell()+1)                             #跳转到下一帧图像
    im.save('frame{:02d}.png'.format(im.tell()))
except:
    print("处理结束")
```

代码采用了 try-except 结构,首先执行 try 中的语句。当图像已经跳转到最后一幅图像帧时,再次跳转(im.tell()+1)会出现异常,此时执行 except 后面的语句,程序结束运行。原始图像如图 3-13 所示,保存的每幅图像帧文件目录结构如图 3-14 所示。其中 fram00.png 是第一帧图像,watson.gif 共有 181 帧图像。

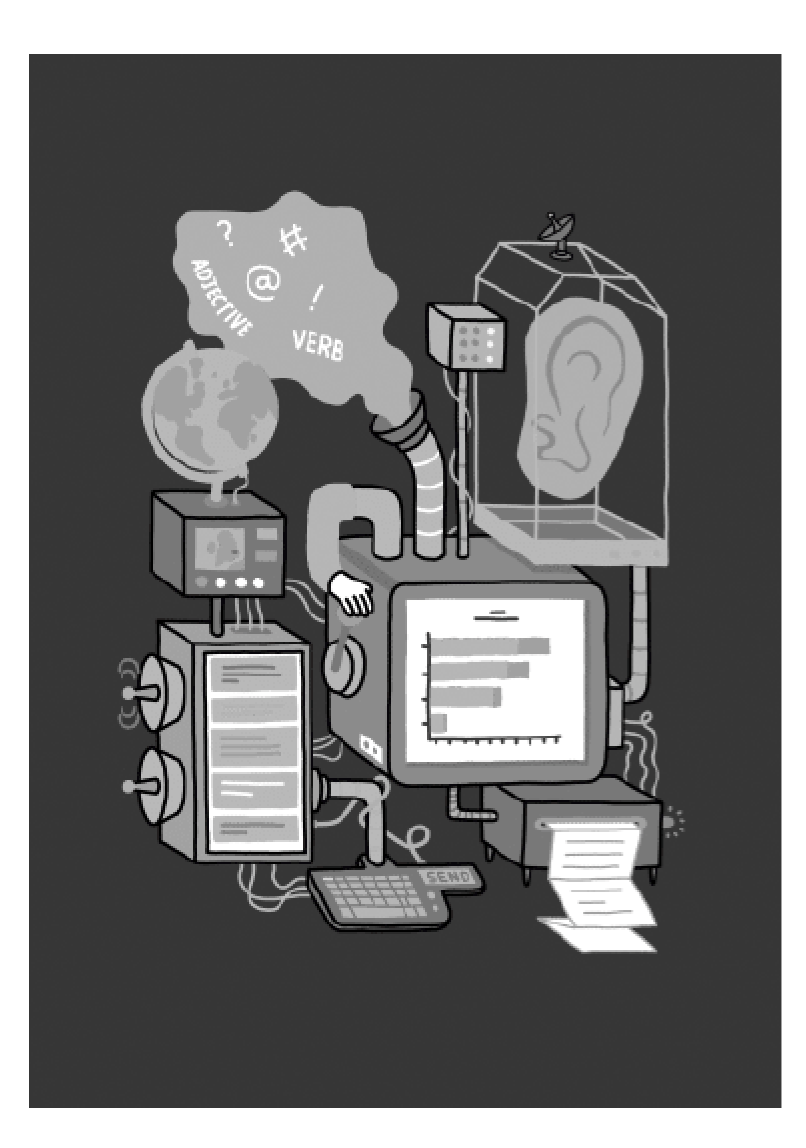



图 3-13　watson.gif

| | | | | | |
|---|---|---|---|---|---|
| frame00 | frame33 | frame66 | frame99 | frame132 | frame165 |
| frame01 | frame34 | frame67 | frame100 | frame133 | frame166 |
| frame02 | frame35 | frame68 | frame101 | frame134 | frame167 |
| frame03 | frame36 | frame69 | frame102 | frame135 | frame168 |
| frame04 | frame37 | frame70 | frame103 | frame136 | frame169 |
| frame05 | frame38 | frame71 | frame104 | frame137 | frame170 |
| frame06 | frame39 | frame72 | frame105 | frame138 | frame171 |
| frame07 | frame40 | frame73 | frame106 | frame139 | frame172 |
| frame08 | frame41 | frame74 | frame107 | frame140 | frame173 |
| frame09 | frame42 | frame75 | frame108 | frame141 | frame174 |
| frame10 | frame43 | frame76 | frame109 | frame142 | frame175 |
| frame11 | frame44 | frame77 | frame110 | frame143 | frame176 |
| frame12 | frame45 | frame78 | frame111 | frame144 | frame177 |
| frame13 | frame46 | frame79 | frame112 | frame145 | frame178 |
| frame14 | frame47 | frame80 | frame113 | frame146 | frame179 |
| frame15 | frame48 | frame81 | frame114 | frame147 | frame180 |
| frame16 | frame49 | frame82 | frame115 | frame148 | 01 |
| frame17 | frame50 | frame83 | frame116 | frame149 | 1 |
| frame18 | frame51 | frame84 | frame117 | frame150 | 2.1DrawPyt |
| frame19 | frame52 | frame85 | frame118 | frame151 | 2 |
| frame20 | frame53 | frame86 | frame119 | frame152 | 3 |
| frame21 | frame54 | frame87 | frame120 | frame153 | 4 |
| frame22 | frame55 | frame88 | frame121 | frame154 | 5.5 |
| frame23 | frame56 | frame89 | frame122 | frame155 | 5 |
| frame24 | frame57 | frame90 | frame123 | frame156 | 6 |
| frame25 | frame58 | frame91 | frame124 | frame157 | 7 |
| frame26 | frame59 | frame92 | frame125 | frame158 | 8 |
| frame27 | frame60 | frame93 | frame126 | frame159 | 9 |
| frame28 | frame61 | frame94 | frame127 | frame160 | 10 |
| frame29 | frame62 | frame95 | frame128 | frame161 | 11 |
| frame30 | frame63 | frame96 | frame129 | frame162 | 21 |
| frame31 | frame64 | frame97 | frame130 | frame163 | 22 |
| frame32 | frame65 | frame98 | frame131 | frame164 | 23 |

图 3-14　图像帧文件目录

## 3.3.3 网络文件的读写

网络时代的大量信息和数据来源于万维网(WWW),由此产生了一系列“网络爬虫”应用。在 Python 中,抓取网页并从中获取数据是非常容易的事情,只需要两个步骤:

(1) 通过网络链接下载网页,需要使用 requests 库。

(2) 解析网页格式,获取其中的数据,需要使用 beautifulsoup4 库。

### 1. requests 库

与 PIL 一样,requests 库是第三方库,必须先安装。安装方法与 PIL 一样,在 DOS 命令行运行“pip install requests”命令。安装成功后,可以用 requests.get()函数下载网页。该函数的参数是要下载的网页地址,用 URL(统一资源定位符)字符串表示。该函数会返回一个 Response 对象。先看下面的代码:

```
import requests
res=requests.get('http://www.nmc.gov.cn/publish/forecast/ABJ/beijing.html')
str=res.text
```

运行结果:

```
>>>type(res)
<class 'requests.models.Response'>
>>>res.status_code==requests.codes.ok
True
>>>type(str)
<class 'str'>
>>>len(str)
67165
```

requests.get()中的 URL 指向的是中央气象台发布的北京市天气预报网页,如图 3-15 所示。res 是返回的 response 对象。查询 res 的 status_code 属性,如果该值等于 requests.codes.ok,说明网页请求成功。下载的网页内容保存在 response 对象的 text 属性中,保存类型是字符串,长度可以用 len()获得。




图 3-15 中央气象台北京市天气预报网页

如果 status. code 属性不等于 requests. code. ok，说明下载失败，此时可以调用 raise_for_status()方法，该方法在网页下载成功时不做任何处理，如果下载出错，就会提示异常，例如，访问一个不存在的网页"http://inventwithpython. com/page_that_does_not_exist"。

```
import requests
res=requests.get('http://inventwithpython.com/page_that_does_not_exist')
```

运行结果：

```
>>>res.raise_for_status()
<Traceback (most recent call last):
  File "<pyshell#8>", line 1, in<module>
res.raise_for_status()
  File "C:\Python36-32\lib\site-packages\requests\models.py", line 929, in raise_for_status
    raise HTTPError(http_error_msg, response=self)
requests.exceptions.HTTPError: 404 Client Error: Not Found for url: http://inventwithpython.com/
page_that_does_not_exist
```

由于这是一个不存在的网页，运行结果会提示 404 状态码，意思是"没找到网页"。使用该方法可以确保程序在下载失败时停止运行。

如果想将下载的网页保存到文件中，要使用 open()和 write()方法。此时必须使用"wb"参数，即"写二进制文件"模式打开文件。用 response 对象的 iter_content()方法可以将网页写入到一个文件，代码如下：

```
import requests
res=requests.get('http://www.nmc.gov.cn/publish/forecast/ABJ/beijing.html')
res.raise_for_status
file=open('weatherBJRead.txt','wb')
for line in res.iter_content():          #以字节形式获取网页内容并写入文件
file.write(line)
file.close()
```

程序以"写二进制文件"模式创建了一个新文件 weatherBJRead. txt。iter_content()方法以字节形式获取网页内容，通过 for 循环将所有字节写入这个文件，其内容如图 3-16 所示。

请注意，iter_content()方法以字节形式获取网页内容，因此在写入文件时必须采用二进制模式打开文件。而上文提到的 response 对象的 text 属性是用 Unicode 编码的文本形式获取网页内容的。如果网页的编码方式不是 Unicode 编码，用 text 获取网页时可能会出现乱码。这时可以找出网页的编码方式，然后用 response 的 encoding 属性来改变

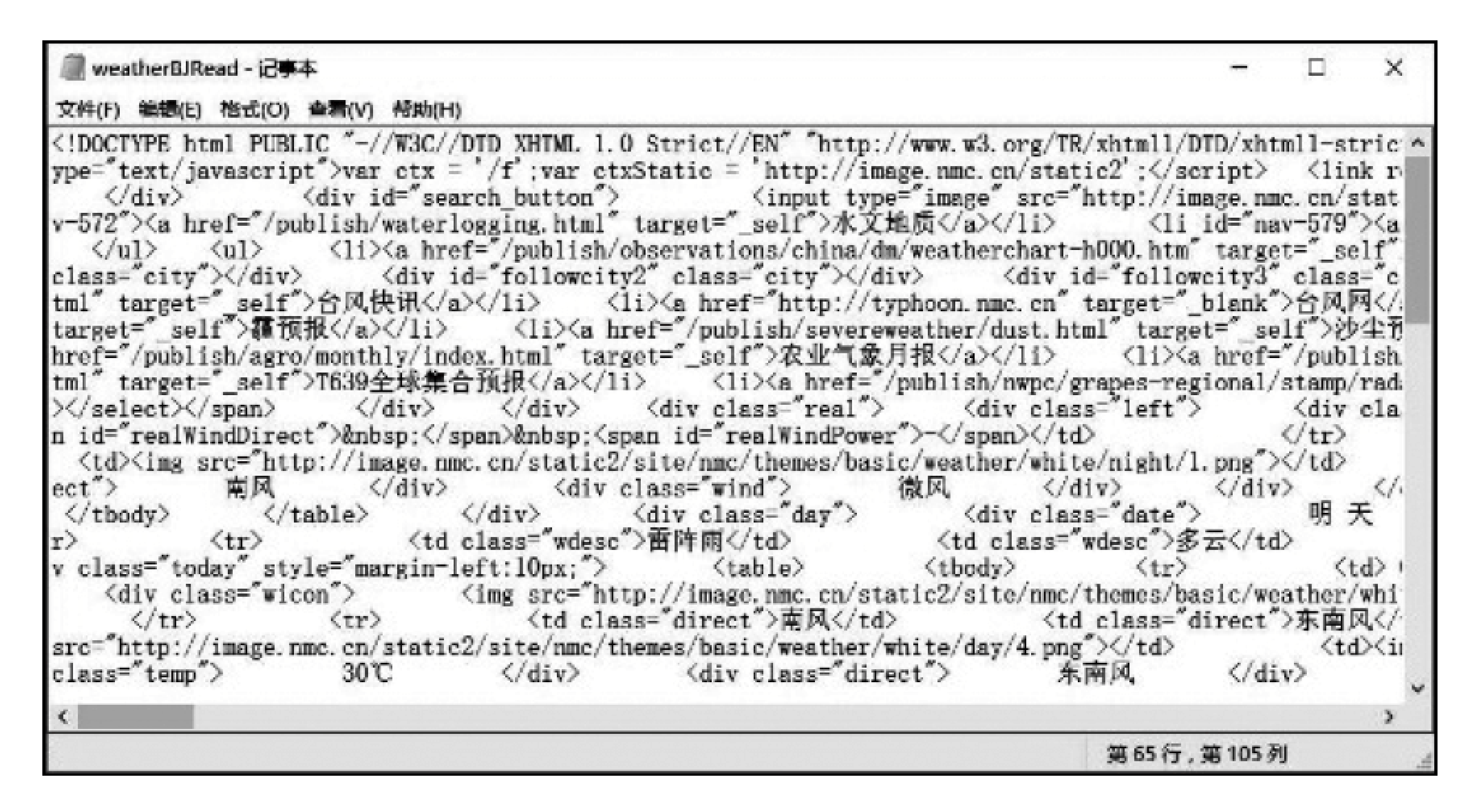



图 3-16　weatherBJRead.txt 文件

编码，例如：

```
res=requests.get('http://www.nmc.gov.cn/publish/forecast/ABJ/beijing.html')
res.encoding='uft-8'
res.encoding='ISO-8859-1'
```

这样就可以正确解析。

## 2. HTML 简介

要想从下载的网页中找到感兴趣的信息，必须了解 HTML 的一些基本知识。

超文本标记语言(Hypertext Marked Language，HTML)是制作超文本的标记语言。超文本是指包含文本、图片、声音、动画、影视等多种多媒体信息的文档。用 HTML 编写的超文本文档称为 HTML 文档，它适用于各种操作系统平台，是万维网(World Wide Web，WWW)的信息表示语言。

HTML 文档是一个纯文本文件，扩展名是.html。文件中的文本被各种“标签”环绕，如<html>与</html>。标签由尖括号包围，里面是标签名称。有些标签成对出现，一个开始标签和一个结束标签包围某段文本，形成一个“元素”。HTML 文档就是由一系列的元素和标签组成的。标签规定了元素的属性和它在文件中的位置，告诉浏览器以怎样的格式显示元素。

HTML 文档包括文档头和文档体两部分。文档头对文档作一些必要的定义，文档体是要显示的页面信息。我们来看一个简单的 HTML 文档——my_web.html。这个 HTML 文档用浏览器打开的效果如图 3-17 所示。

因为 HTML 文档是纯文本文件，可以在记事本中进行编辑，在保存文件时扩展名定义为“.html”，就会生成 HTML 文档。这个文档的代码如下：




图 3-17　my_web 网页

```
<html>
<head>
<title>一个简单 html 文档</title>
</head>
<body>
<center>
<h1>欢迎光临这个主页</h1>
<br>
<hr>
<font size=7 color=red>
这是我的编程语言学习主页
</font>
</center>
<font size=5 color=blue>
<pid='python'>学习 python,请访问<a href="http://inventwithpython.com">python 编程</a></
    p>
<pid='java'>学习 java,请访问<a href="http://www.w3school.com.cn/js/index.asp">java 编程</a
    ></p>
</font>
</body>
</html>
```

其中，<html>与</html>是最外层的一对标签，HTML 文档中的所有文本和标签都包含在其中，它表示该文档是用 HTML 编写的。<head>与</head>是 HTML 文档的头部标签，在浏览器窗口中，头部信息不会显示在正文，在这对标签中可以插入其他标记，用以说明文件的标题和整个文件的一些公共属性。若不需头部信息则可省略此标记。<title>与</title>是嵌套在<head>头部标签中的，标签之间的文本是文档标题，它会显示在浏览器窗口的标题栏。<body>与</body>标签之间的文本是要显示在浏览器上的页面内容。上述几对标签在文档中都是唯一的，head 标签和 body 标签嵌套在 HTML 标签中。<center>与</center>是居中对齐标签。在需要居中的内容开头处加<center>，结尾处加</center>。<br>是换行标签，当文件显示在浏览器上时，该标签之后的内容将在下一行显示。<hr>是水平分割线标签，用于段落与段落之间的分

隔，使文档结构清晰、文字编排更整齐。<font>与</font>是文字格式控制标签，用于控制文字的字体、大小和颜色。设置格式示例如下：

```
<font face=值 1 size=值 2 color=值 3>文字</font>
```

其中 face 设置文字使用的字体，默认值为宋体；size 设置文字的大小，默认值是 3；color 设置文字的颜色，默认值是黑色。如果用户系统中没有 face 属性所指的字体，则使用默认字体。

<p>与</p>是段落标签，表示一个段落。<a>与</a>标签定义超链接，用于从一个页面链接到另一个页面。<a>标签最重要的属性是 href 属性，它指定链接网页的 URL 地址。在浏览器中，未被访问的链接带有下画线并且默认为蓝色；已被访问的链接带有下画线并且默认为紫色；活动链接带有下画线并且默认为红色。有时某些标签具有 id 属性，用来在页面上唯一地确定该元素，例如代码中的两个链接地址。

HTML 的标签远不止这些，感兴趣的读者可以更深入地了解 HTML 语法。

### 3. BeautifulSoup 库

了解 HTML 的基本知识后就可以尝试解析网页。例如，要想从上面的网页 my_web.html 中获取编程语言学习网站的链接地址，首先打开网页，然后在页面上右击，在弹出的菜单中选择“查看网页源代码”，显示该网页的 HTML 代码，如图 3-18 所示。

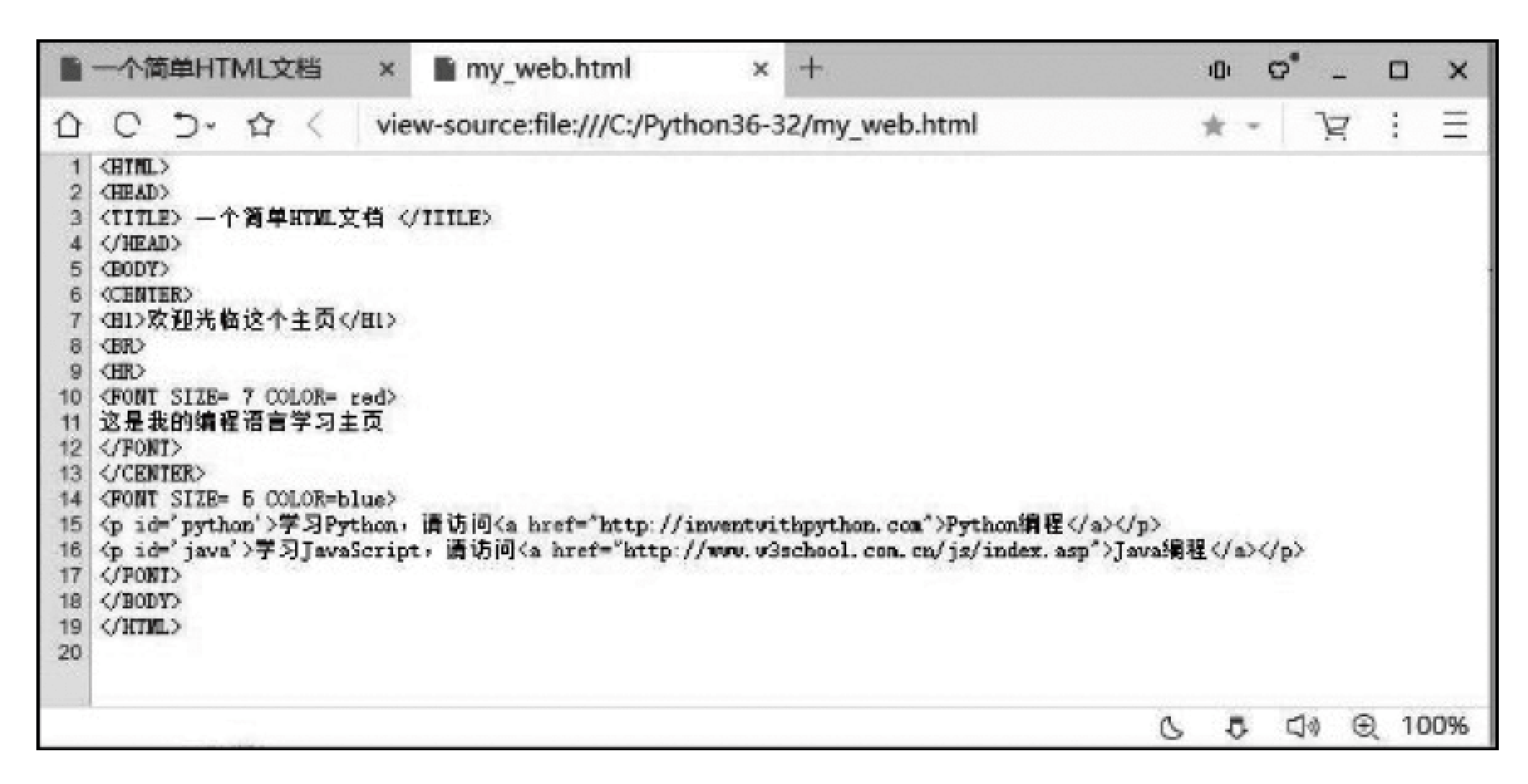



图 3-18 my_web 网页源代码

观察代码可以发现，网页中表示链接地址的部分是<ahref="http://inventwithpython.com">和<ahref="http://www.w3school.com.cn/js/index.asp">。我们需要从<a>标签中提取网站链接地址，此时可以用 bs4 库帮忙。

BeautifulSoup4 库，简称 bs4 库，用于从 HTML 页面中提取信息。安装方法与 requests 相同，在 DOS 命令行运行“pip install beautifulsoup4”。注意导入这个库时要使用 import bs4。将待解析的 HTML 文档传递给 BeautifulSoup()函数，可以返回一个 BeautifulSoup 对象，它包含了 HTML 页面中的所有标签，如<head>、<body>等。这

些标签作为 BeautifulSoup 对象的属性，可以用“<BeautifulSoup 对象>. 标签名称”的形式直接获得，例如：

```
from bs4 import BeautifulSoup
res=open('my_web.html','r',encoding='utf-16LE').read()
soup=BeautifulSoup(res)
```

运行结果：

```
>>> soup.head
<head>
<title>一个简单 HTML 文档</title>
</head>
>>> soup.p
<p id="python">学习 Python,请访问<a href="http://inventwithpython.com">
  Python 编程</a></p>
>>> soup.a
<a href="http://inventwithpython.com">Python 编程</a>
```

因为网页已经保存在当前目录下，不需要使用 requests 下载，直接用 open()打开后读取内容并保存为字符串 res。soup 是 BeautifulSoup 对象，可以返回多个标签属性，这些标签也称为标签对象，其结构基本类似，例如，标签对象<a>的结构如下：

```
<a href="http://inventwithpython.com">Python 编程</a>
```

标签对象包含 name、attrs、contents 和 string 属性，其含义如表 3-2 所示。尖括号(<与>)里的标签名称“a”是 name 属性，其他项如“href”是 attrs 属性。尖括号之间的内容“Python 编程”是 string 属性。

**表 3-2　标签对象的属性**

| 序号 | 属　性 | 含　　义 |
|---|---|---|
| ① | name | 标签名称，例如“a”，字符串类型 |
| ② | attrs | 标签对象的属性，例如“href”，字典类型 |
| ③ | contents | 该标签下所有子标签的内容，列表类型 |
| ④ | string | 标签包围的文本，网页上显示的文字，字符串类型 |

通过查询标签对象的属性就可以获得需要的信息。由于标签中可以嵌套其他标签，因此返回 string 属性时要注意：

① 如果标签内部没有其他标签，string 返回其中内容。

② 如果标签内部有且只有一个标签，string 返回最里面标签的内容。

③ 如果标签内部有超过一层嵌套的标签，string 返回 None。

HTML 文档中同一个标签会有很多内容，例如 my_web. html 中各有两处<a>标签和<p>标签，直接调用 soup. a 和 soup. p 会返回第一个标签内容。如果要查找同一个标签的其他内容，可以使用 BeautifulSoup 的 find()和 find_all()方法。这两个方法会遍历整个 HTML 文档，按照参数返回标签内容。用法如下：

```
BeautifulSoup.find_all(name,attrs,recursive,string,limit)
```

其中 recursive 参数设置查找层次，只查找当前标签下一层时使用 recursive＝False。limit 设置返回查找结果个数，默认返回全部结果。find()方法与 find_all()类似，区别是前者以字符串形式返回找到的第一个结果，后者以列表类型返回全部结果。

例如，要获得编程语言学习网站的链接地址，需要获得标签对象<a>的 href 属性，可以用如下代码：

```
lis=soup.find_all('a')  #查找所有标签<a>
urllist=[]
for i in range(len(lis)):
urllist.append(lis[i].attrs['href'])
```

运行结果：

```
>>>urllist
['http://inventwithpython.com', 'http://www.w3school.com.cn/js/index.asp']
```

程序中的列表 lis 返回了 HTML 网页中所有的<a>标签对象，然后借助 for 循环结构取出每个<a>标签对象中的“href”属性值，即网站链接地址，再将它添加到新列表 urllist 中。

# 本 章 小 结

(1) 了解文件的功能。
(2) 理解文件类型，掌握文件名定义方法，掌握文件位置的描述方法。
(3) 掌握文件的打开和关闭方法。
(4) 掌握文件的读写基本操作。
(5) 掌握第三方库的安装方法。
(6) 了解 PIL 库，熟悉图像文件的基本读写和常用处理方法。
(7) 了解 requests 库和 bs4 库，熟悉网络文件的下载和解析方法。

# 习　　题

## 1．选择题

(1) 下列操作能够创建文件对象的是【　　】。

A) open　　B) file　　C) create　　D) make

(2) 下列操作不能够读取文件的是【　　】。

A) read　　B) readline　　C) readlines　　D) readall

(3) 关于语句 f=open("a.txt", "w+"),下列说法正确的是【　　】。

A) 只能读取数据　　B) 只能写入数据

C) 文件必须已经存在　　D) 文件可以不存在

(4) 下列程序的输出结果是【　　】。

```
f=open("w.txt","w+ ")
f.write("Lux et Veritas")
f.seek(7)
s=f.read(3)
f.close()
print(s)
```

A) eri　　B) Ver　　C) tas　　D) Lux

(5) 下列程序的输出结果是【　　】。

```
f=open("w.txt","w")
f.write("Lux et Veritas")
f.close()
f=open("w.txt","r")
f.read(3)
f.seek(4,1)
```

A) 5　　B) 6　　C) 7　　D) 8

(6) 执行下列语句会报错,错误在第【　　】行。

```
f=open("w.txt","w+ ")
f.write("Lux et Veritas")
f.seek(4,1)
```

A) 1　　B) 2　　C) 3　　D) 并不会报错

## 2．程序设计题

（1）编写程序，让用户输入自己的姓名、年龄、最喜欢的颜色和最喜欢的书。程序将这四项内容保存在一个文本文件中，每一项内容分别放在单独的一行上，再逐行显示文件的内容。

（2）编写程序自动造句，每个句子包含四个部分：形容词＋名词＋动词＋名词。例如：聪明的阿凡提喜欢骑毛驴（提示：这些词可以事先保存在4个不同的文件中，程序随机选择一个词组成句子）。

（3）Python 学习笔记：在文本编辑器中创建一个新文件，写几句话总结你至今学到的 Python 知识，其中每句话占一行，内容是“我可以用 Python……”。保存这个文件。编写程序，它读取该文件并打印三次：第一次打印时读取整个文件；第二次打印时遍历文件对象；第三次逐行读取内容，并用方法 replace()将“Python”替换成另外一门编程语言的名称，如 C、Java 等，将修改后的各行都打印到屏幕上。

（4）访客名单：编写一个 while 循环，提示用户输入其名字。用户输入其名字后，在屏幕上打印一句问候语，并将一条问候记录添加到文件 guest_book. txt 中。要求每条记录独占一行。

# 第4章 程序结构设计

## 4.1 程序的基本结构

### 4.1.1 Python 程序结构概述

一个典型的 Python 程序可视为一个模块的系统，其程序架构见图 4-1。它有一个顶层文件（启动后可运行程序）以及多个模块文件（用来导入工具库）。

更细致地来讲，顶层文件、模块文件和程序的相互关系可归纳为以下几点。

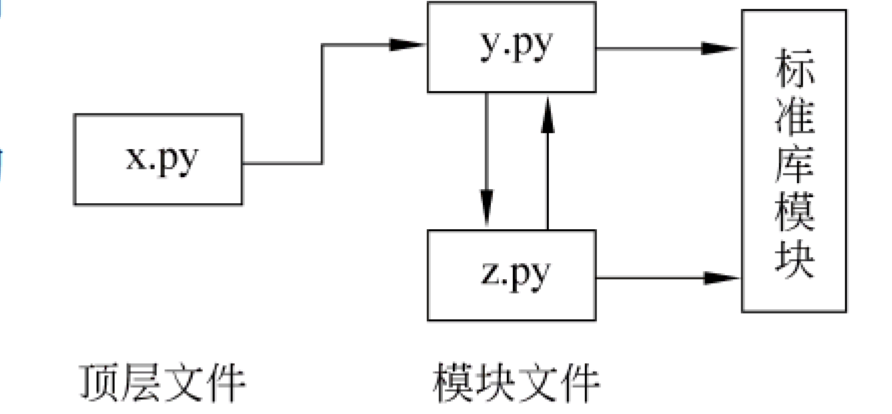



图 4-1 Python 程序架构

**1. 程序和模块的相互关系**

Python 环境中，程序是作为一个顶层的文件来定义的，配合多个模块文件的支持。模块文件在程序设计中又简称为“模块”。Python 程序设计中，设计每个模块又常常涉及若干标准库模块。Python 语言提供了庞大的标准库模块，包括人工智能、大数据分析与处理、数据可视化、人脸识别等标准库。

**2. 顶层文件的含义**

这一文件包括程序的主要控制流程，也就是用来启动应用的相关文件。顶层文件就像一名“指挥官”，由它来协调和控制各个模块，保证程序的正常运行。

**3. 模块文件的含义**

模块文件可视为顶层文件下的“士兵”或者“工具仓库”，这些士兵或者工具是被顶层文件所使用的组件。顶层文件使用了模块所提供的士兵或工具，此外模块文件也会根据权限使用其他模块所定义的士兵或工具。

**4. 模块执行环境**

一个典型的模块文件主要包含变量、函数、类以及其他的模块，此外函数也有自己的本地变量。

### 4.1.2 算法概述

计算机程序可视为数据结构和算法的集成，这种集成体现为以下两方面的内容。

#### 1. 针对数据的描述

数据是程序实现的基础，在程序中要指定数据的类型和数据的组织形式，即数据结构(Data Structure)。

#### 2. 针对数据的具体操作

这一具体操作过程，也就是算法，算法要依靠程序来完成功能，程序需要算法作为灵魂。

以上两方面内容体现了数据结构和算法对程序设计的重要性。实际上，一个程序除了数据结构和算法两个核心要素之外，还应当采用某种程序设计方法进行程序设计，并且应用某一类计算机语言加以描述。因此，一个程序可以表示为“程序＝算法＋数据结构＋程序设计方法＋编程语言和环境”。

从程序设计的角度而言，算法是灵魂，数据结构是加工对象，编程语言是工具，此外编程要采用合适的方法。算法是解决“做什么”和“怎么做”的关键问题。程序的操作语句就是算法的具体体现。算法不仅仅体现在数值计算等学科领域，“算法”也时时刻刻存在于我们的生活中。例如，你准备去某个国家参加一次国际学术会议，首先你要办理签证手续，然后购买机票(或者火车票)和预订酒店等，接下来去该国家参加会议。要上大学，首先要满足报名条件，接下来填报名单、交报名费、取准考证、按时参加考试、拿到通知书、到学校报到等。以上两个案例都是按照一定的顺序进行、缺一不可，且要保证每个步骤的次序，这些步骤就是“算法”的具体化描述。

此外，对于同一个问题，可以有不同的解决方法和步骤。例如，求自然数 1 到 100 之间所有数的和(包含 1 和 100)，可采取先进行 1＋2，再加 3，再加 4，一直加到 100；也可以采取如下方式。

```
(100+1)+…+(50+5)=101×5=5050
```

对于这一问题，还有其他的解决方案。一般而言，不同的方案需要的步骤有所不同，这种步骤的差异就体现在方法的运算效率，也就是方法的计算复杂度。

### 4.1.3 算法的特点

鉴于算法在程序中的核心地位，算法的特性如下。

#### 1. 有穷性

一个算法应包含有限的操作步骤，而不可能是无限的。事实上，“有穷性”表示“运算

的步骤在合理的范围之内”。如果让计算机执行一个上百年才能结束的算法,虽然是有穷的,但已大大地超出了合理的限度,也不能视为有效算法。但“合理的范围”并无严格的标准,往往依赖于问题的背景以及所处的计算机环境。

### 2. 确定性

算法中的每一个步骤都应当是确定的,而不应当是含糊的、模棱两可的。例如,我们生活中讲“小明是一个乖孩子”,这种描述就是模糊的,因为“乖”是一个什么样的程度,我们无法具体量化,每个人的标准都有差异。另外我们判断闰年,如果将条件表述为“能被一个整数整除”,这也是“模糊的”,应该说明具体是哪个整数。换句话说,算法的含义应当是明确的,不应当产生歧义。

### 3. 零个或多个输入

程序的输入是指在执行算法时需要从外界所获得的必要的数据信息。例如,在执行等差数列求和,需要输入首项、公差和项数。有些算法是没有输入的,比如我们通过程序直接输出一个字符串。

### 4. 一个或多个输出

所有算法的目的是为了求解,“解”就是程序的输出。例如,通过冒泡排序算法对 5、3、2、6、7 进行从小到大输出。需要注意的是,算法的输出不一定就是计算机的打印输出,一个算法得到结果就表示算法实现了输出。没有输出的算法是没有任何意义的。

### 5. 有效性

就一个算法而言,每一个步骤都应当能有效地执行,并得到相应确定的结果。例如,对于两个数 x 和 y,若 y=0,则在 Python 中,x/y 是不能有效执行的。

因此在程序设计过程中,要遵循以上 5 个特征,才能设计出可执行的有效程序,进而解决具体问题。

## 4.1.4 算法的表示

算法的表示基本可以分为三类:自然语言表示算法、程序流程图、程序代码。上述三种表示是一个逐渐深入的过程。

### 1. 自然语言描述算法

自然语言就是人们在日常生活中使用的语言,可以是汉语或者其他语言。用自然语言描述的优势就是通俗易懂,但文字冗长,往往容易出现歧义。自然语言的本质决定了其含义往往不严格,需要综合考虑上下文的语义关系,才能正确地理解其含义。其中经常应用的一种自然语言描述就是“输入(Input)、处理(Processing)和输出(Output)”模式,简称为 IPO 模式。

### 2. 程序流程图

程序流程图就是采用一些图框来表示算法的各种操作。用图形来描述算法，直观形象、便于理解。美国国家标准化协会制定了一些常用流程图符号(图 4-2)，目前该体系已被世界各国程序开发者普遍采用。

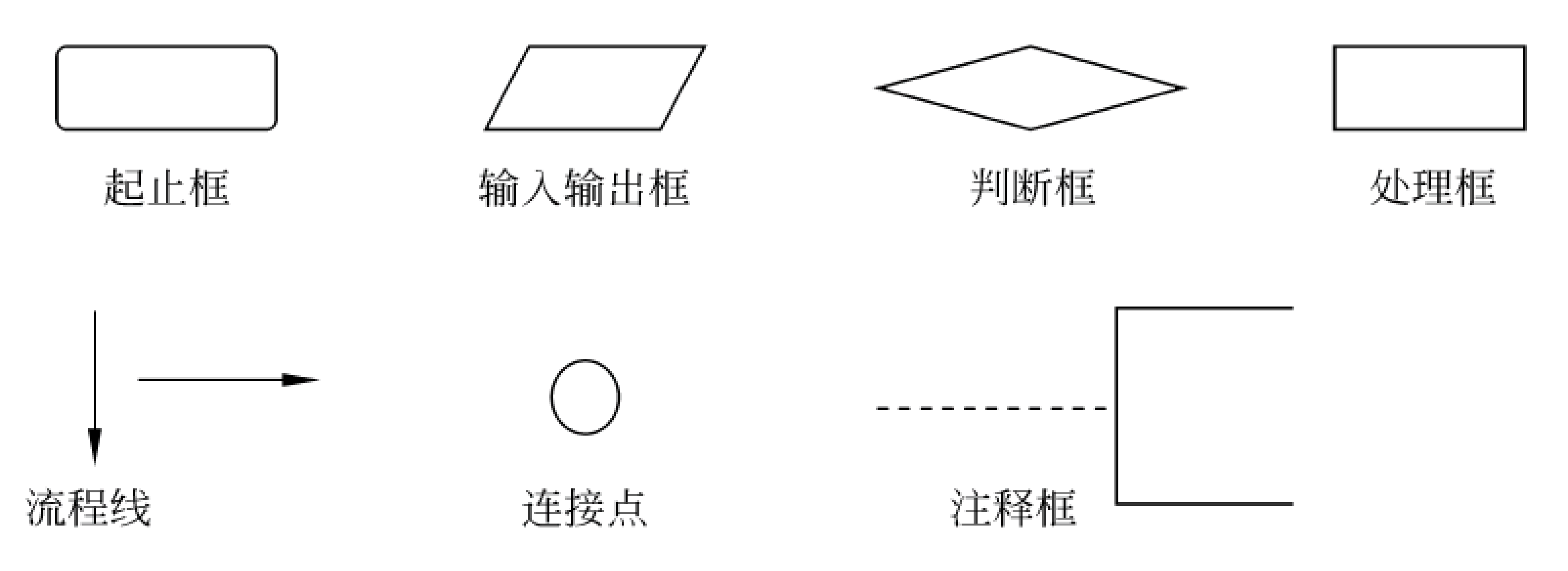



图 4-2　程序流程图常用符号

起止框表示一个程序的开始和结束；输入输出框表示数据输入或者结果输出；判断框表示一个条件是否成立；处理框表示一组处理过程；流程线表示程序的执行路径；连接点是用于将画在不同地方的流程线连接起来；注释框是为了对流程图中某些操作给出必要的补充说明，从而帮助阅读流程图的人更好地理解流程图的含义。

### 3. 程序代码

程序代码就是在各类编译环境下可执行的程序代码，也就是最终人们需要熟练掌握的必要工具。下面通过几个算法的例子，来说明以上三者(IPO、程序流程图和程序代码)的差异性和关联性。

**【例 4-1】** 求圆面积的两种表示，如图 4-3 所示。

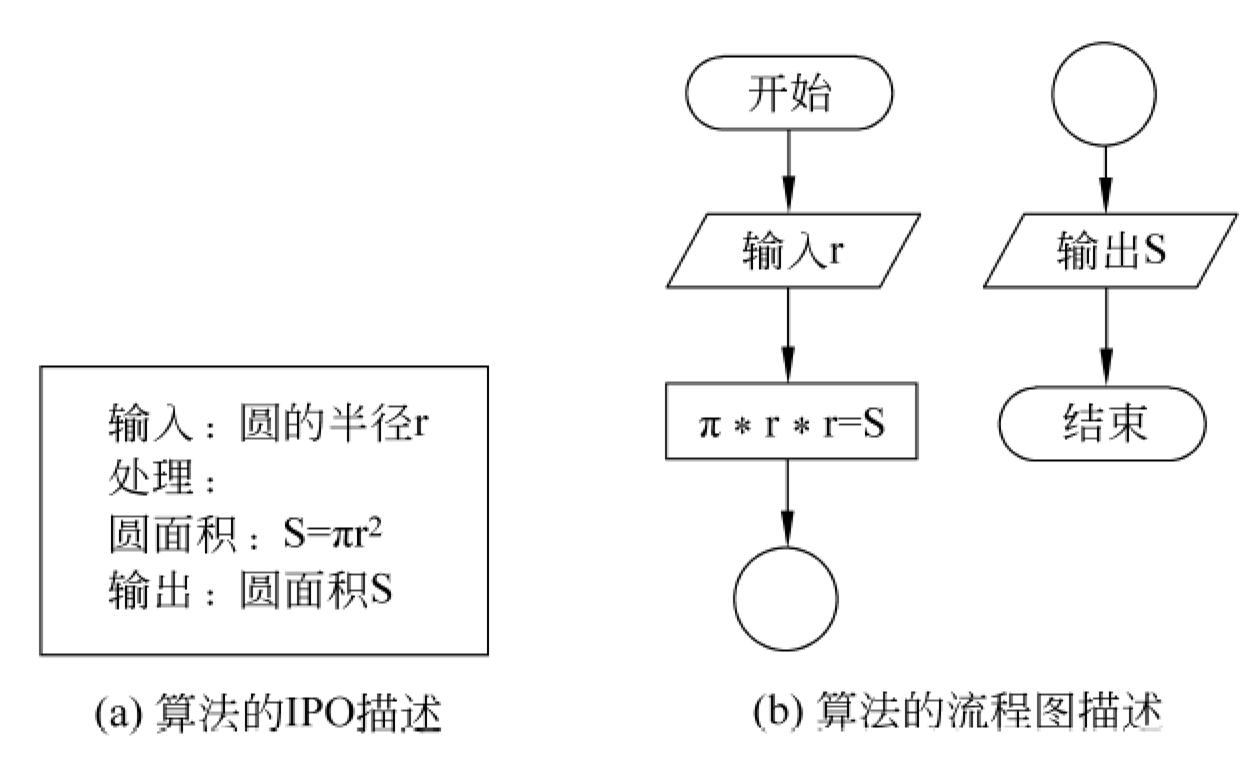


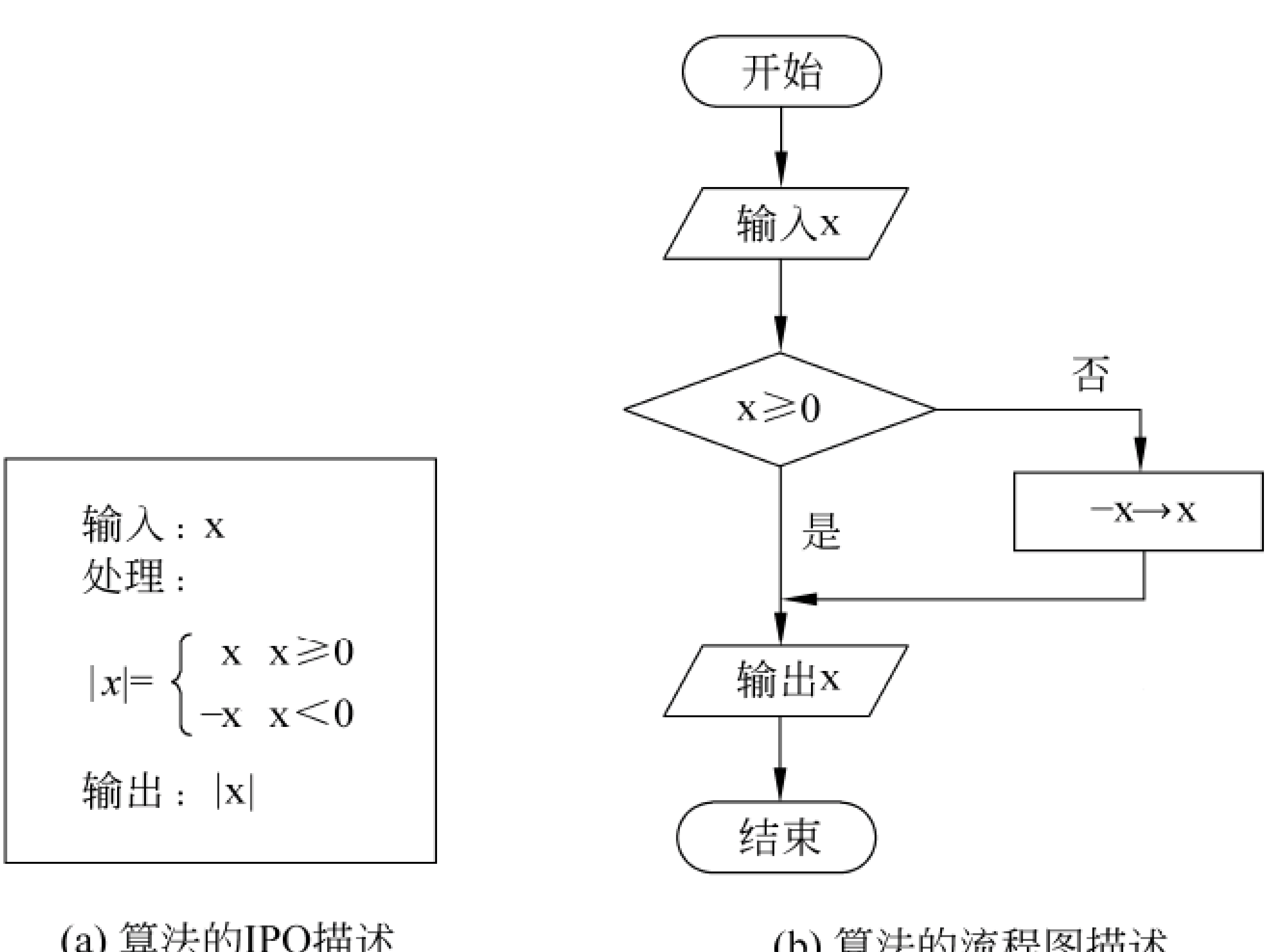

图 4-3　求圆面积的两种算法描述对比(IPO 和流程图)

```
r=eval(input("输入圆的半径 r: "))
S=3.14*r*r
print("面积: ",S)
```

运行结果：

```
输入圆的半径 r: 12
面积: 452.15999999999997
```

**【例 4-2】** 函数 f(x)=|x|的两种表示，如图 4-4 所示。




(a) 算法的IPO描述　　(b) 算法的流程图描述

图 4-4　f(x)=|x|函数的两种描述方法(IPO 和流程图)

```
x=eval(input("输入实数:"))
if x>=0:
    print("x的绝对值为",x)
else:
    print("x的绝对值为",-x)
```

运行结果：

```
输入实数:-6
x的绝对值为 6
```

**【例 4-3】** $\sum_{i=1}^{100} i$ 的两种表示，如图 4-5 所示。

```
S=0
i=1
while i<=100:
    S=S+i
    i+=1
print(S)
```

运行结果：

```
5050
```

通过以上三个例子可以看出，IPO方式通俗易懂，但是不能体现算法的过程；流程图表示算法直观形象，能够较为清楚地显示出各个框之间的逻辑关系。流程图的主要局限性在于占用篇幅较多，且当算法比较复杂的话，设计流程图既浪费时间也不太方便。但每一个程序设计人员都应当熟练掌握流程图，会看也会画。Python代码则能直观地输出算法的执行结果。

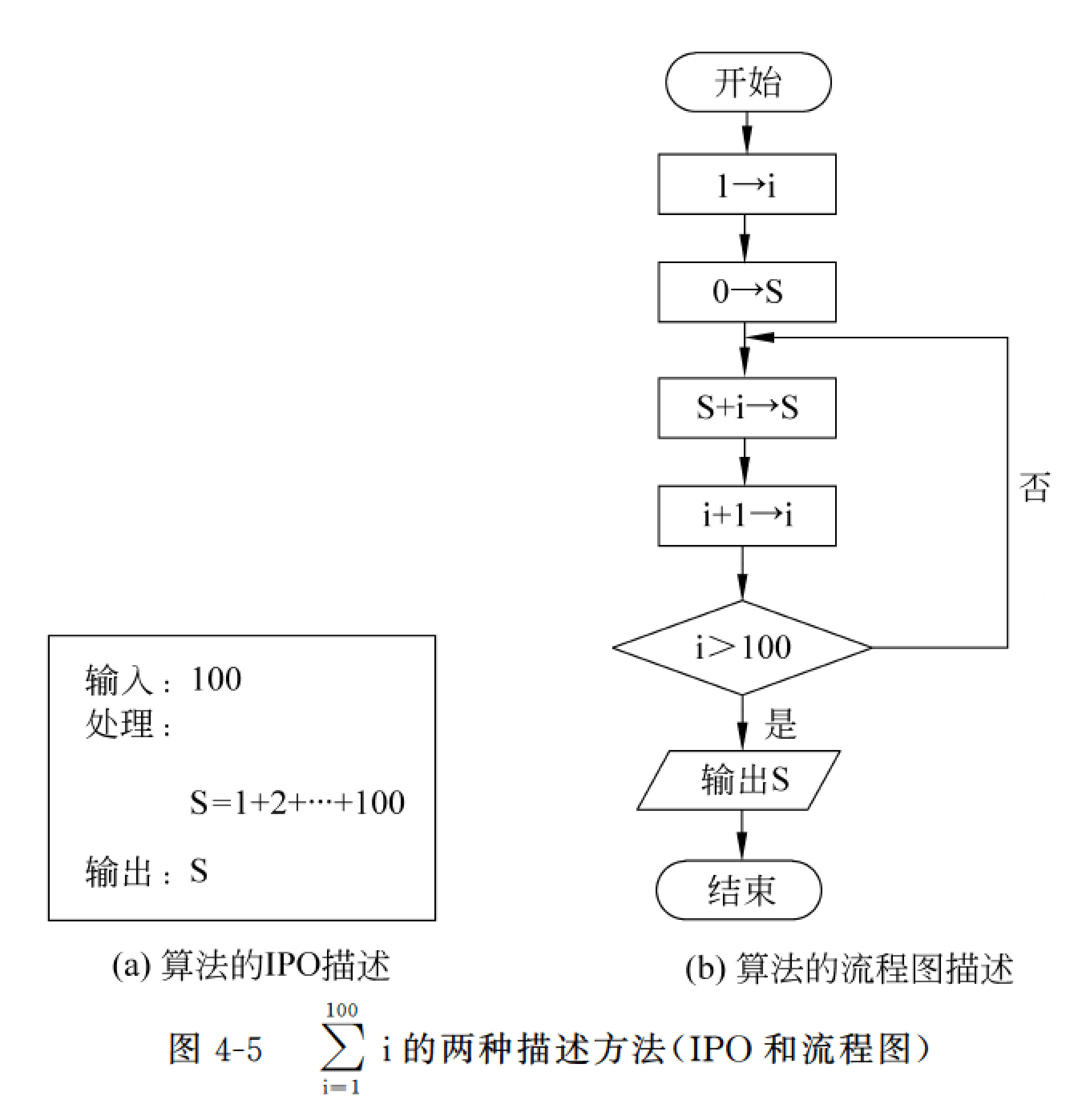



图4-5 $\sum_{i=1}^{100} i$ 的两种描述方法（IPO和流程图）

## 4.1.5 程序的三种基本结构

Bohra和Jacopini在1966年提出了程序设计的三种基本结构：顺序结构、选择结构和循环结构。这三种结构可以很好地描述一个良好算法的基本单元。

### 1. 顺序结构

如图4-6所示，虚线框内是一个顺序结构，A和B两个框是顺序执行的，即在执行完A框所指定的各类操作后，必然执行B框所指定的操作。顺序结构是最简单也是最基本的一种结构。例4-1就是一个典型的顺序结构。

### 2. 选择结构

如图4-7所示，虚线框内是一个选择结构。此结构必须包含一个判断框。根据给

定的条件 p 是否成立而选择执行 A 框或者 B 框。就例 4-2 而言，其判断条件 p 定义为“x≥0”。

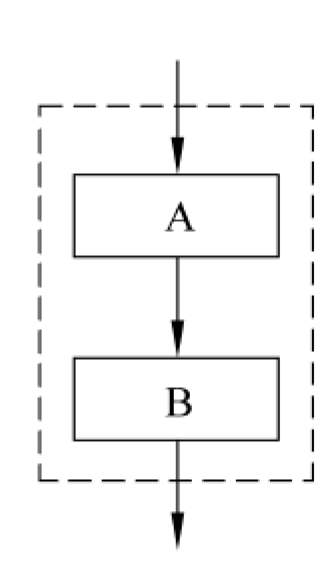




图 4-6　流程图表示的顺序结构图

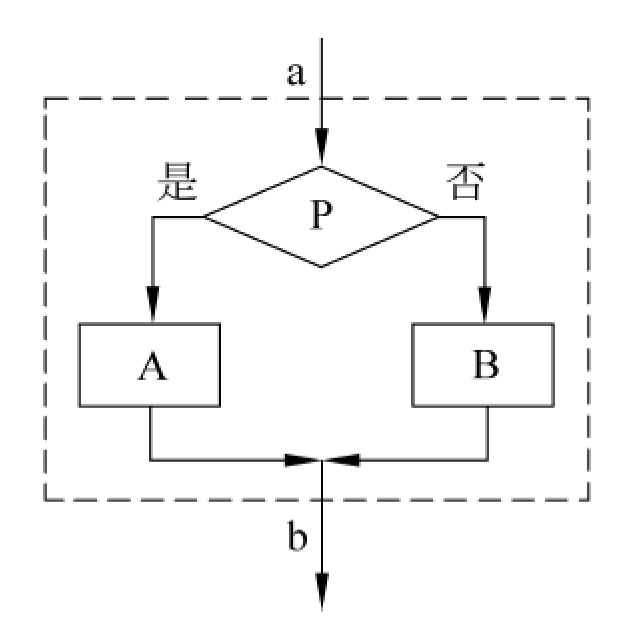



图 4-7　流程图表示的选择结构图

### 3. 循环结构

如图 4-8 所示，循环结构即反复执行某一部分的操作，有两类结构：for 语句和 while 语句。

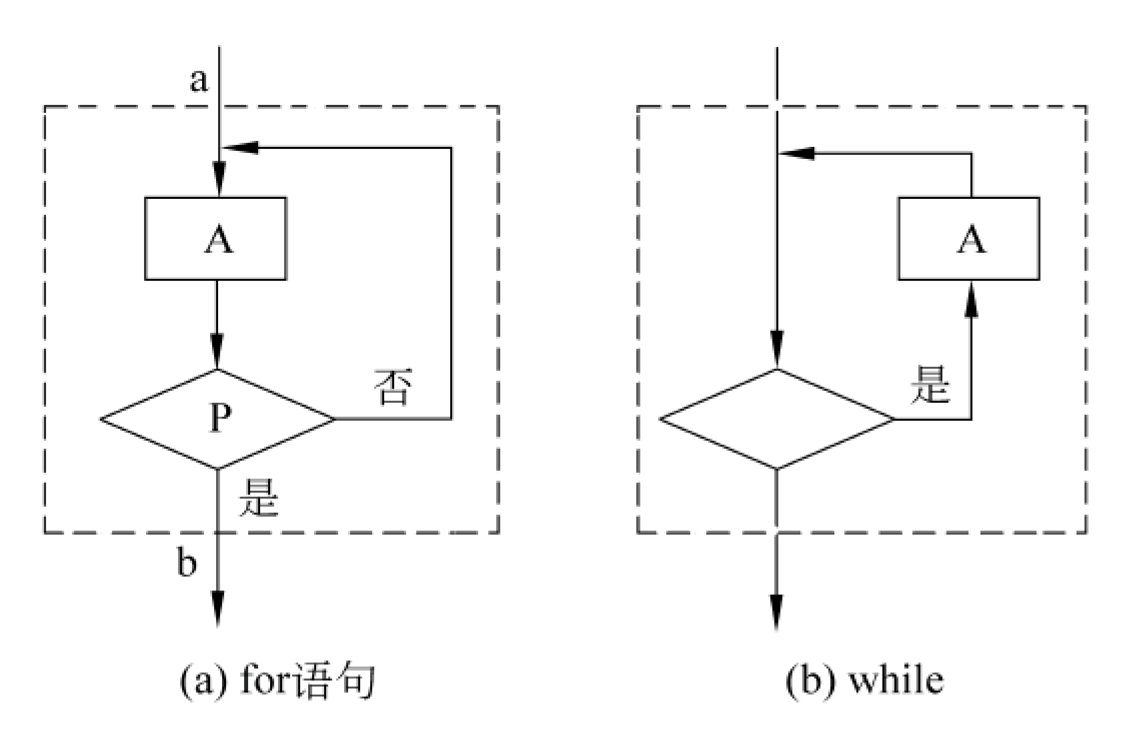



图 4-8　流程图表示的循环结构图

归纳一下以上三种基本类型的结构，其共同点如下：

（1）只有一个入口。图 4-6～图 4-8 中的 a 点为入口点。

（2）只有一个出口。图 4-6～图 4-8 中的 b 点为出口点。需要注意的是，判断框有两个出口，而选择结构只有一个出口。请勿将菱形判断框的出口和选择结构的出口相互混淆。

（3）结构内的每一个部分都可能被执行到。换句话说，对每一个框来说，都应当有一条从入口到出口的路径。

（4）结构内不存在死循环。

已经证明，上述三种基本结构可描述任何复杂的问题。由基本结构构成的算法属于“结构化”的算法，它不存在无规则的转向，只在其基本结构内才允许相应的分支和向前（或向后）的跳转。程序设计发展到现在，基本结构不仅仅局限于上述三种，只要符合上述 4 点的结构都可视为基本结构。因此，程序设计人员可以自定义基本结构，并通过这些基

本结构组成结构化程序。

以上述三种结构为基础，I. Nassi 和 B. Shneiderman 在 1973 年引入了 N-S 流程图（N 和 S 是这两位学者英文姓的首字母），完全去掉了带箭头的流程线。全部算法写在一个矩形框内，该框内还包含一系列子框，用以描述从属或者流程。N-S 流程图主要的符号定义如下：

（1）N-S 的顺序结构。如图 4-9，X 和 Y 两个框组成一个顺序结构。

（2）N-S 的选择结构。如图 4-10，当 P 条件成立（True）执行 X 操作，反之执行 Y 操作。

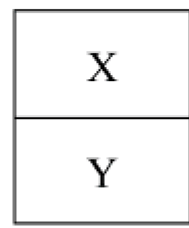
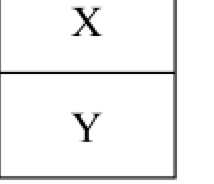




图 4-9　N-S 表示的顺序结构图

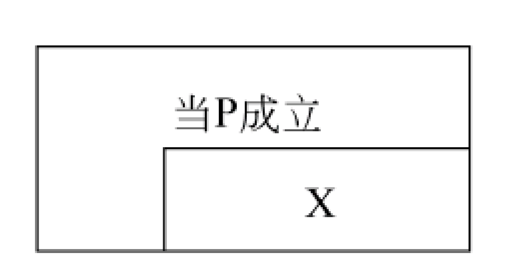


图 4-10　N-S 流程图的选择结构图

（3）N-S 的循环结构。如图 4-11(a)和图 4-11(b)，其中图 4-11(a)表示当 p 条件成立时反复执行 X 操作，直到 p 条件不成立为止；图 4-11(b)表示先执行 X 操作，直到 p 条件不成立为止。

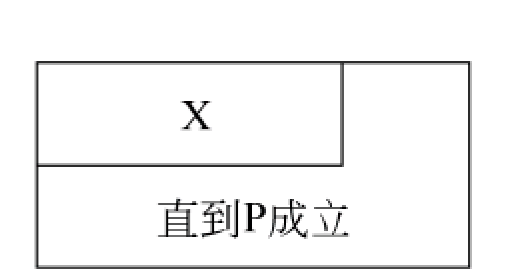
为了更好地了解 N-S 流程图，下面给出三个例子加以解释。

**【例 4-4】**　重新使用 N-S 流程图表示例 4-1 的计算圆面积算法，如图 4-12 所示。










(b) 先循环再判断结构

图 4-11　N-S 流程图的循环结构图

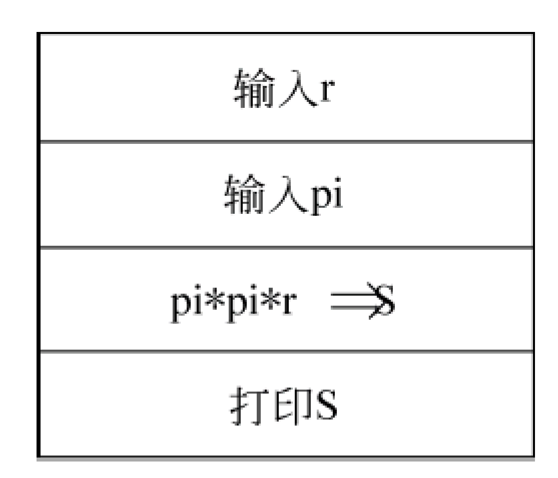



图 4-12　N-S 流程图表示示例一

**【例 4-5】**　重新使用 N-S 流程图表示例 4-2 的函数 $f(x)=|x|$，如图 4-13 所示。

**【例 4-6】**　重新使用 N-S 流程图表示例 4-3 的 $\sum_{i=1}^{100} i$，如图 4-14 所示。

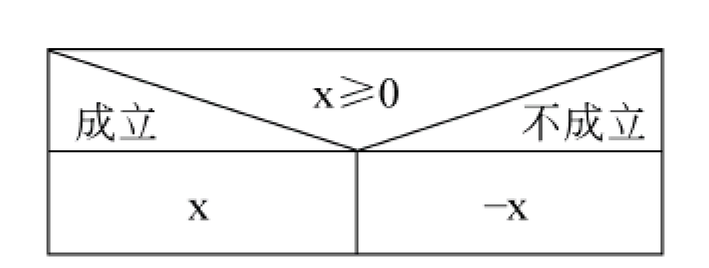



图 4-13　N-S 流程图表示示例二

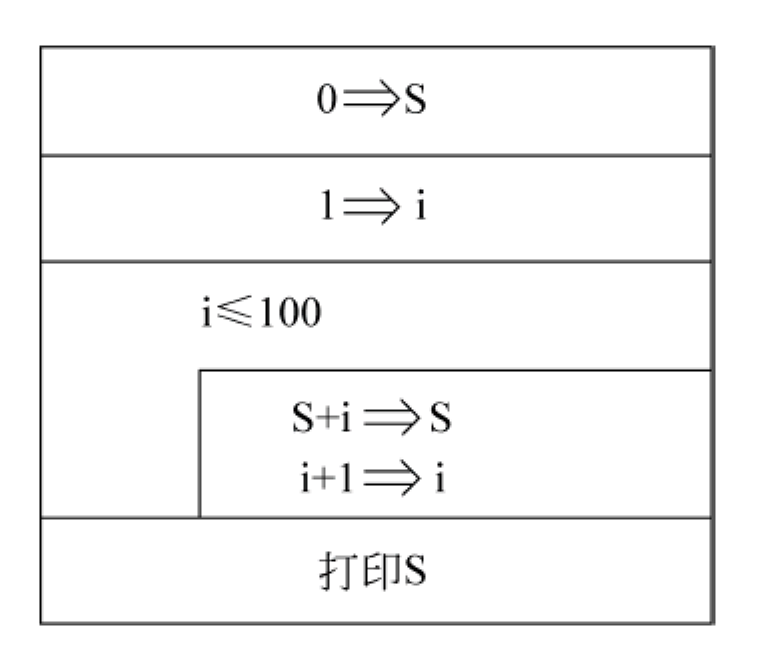



图 4-14　N-S 流程图表示示例三

# 4.2　程序设计中的表达式

## 4.2.1　Python 语言的关系表达式

在程序设计中，关系运算符和表达式扮演着非常重要的角色。其实“关系运算”就等价于“比较运算”。将两个值进行比较，判断其结果是否符合程序给定的条件。例如，x>5 是一个关系表达式，“>”就是一个关系运算符，如果 x 的值为 6，则满足给定的“x>5”条件，关系表达式的值为“True”（即条件为真）；如果 x 的值为 3，则不满足给定的“x>5”条件，关系表达式的值为“False”（即条件为假）。

用关系运算符将两个表达式（字符表达式、算术表达式、关系表达式、逻辑表达式、赋值表达式）连接起来的式子，称之为关系表达式。

关系表达式的值是一个逻辑值，即“True”和“False”。例如，“'python'>'Python'”的值为“True”。

## 4.2.2　Python 语言的逻辑表达式

用逻辑运算符将关系表达式或逻辑量连接起来的式子就称为逻辑表达式。例如，有以下形式的逻辑表达式“a>b && x>y”。如果 a>b 和 x>y 同时成立，则该逻辑表达式为“真”，值等于 1。接下来，介绍 Python 语言中的逻辑运算符和逻辑运算。

基于逻辑运算符连接的式子，称为逻辑表达式。根据逻辑运算符的运算规则，其表达式的值应该是一个逻辑量“True”或“False”，但也要注意到，在 Python 语言中，非“0”的数值或者字符串也可以定义逻辑表达式，其结果并不是“True”或“False”，而是相应的数值或者字符串，这是因为在 Python 语言中，非“0”值等价于“True”，请参考表 4-1。

**表 4-1　逻辑表达式及运算结果**

| 示　　例 | | 示例说明 |
|---|---|---|
| not 0 | True | 在使用逻辑运算符时，最需要注意的一点是短路逻辑（或叫做惰性求值），它表达的意思是：逻辑运算是自左向右进行的，如果左边已经决定结果，就不会再去做右边的计算。此外，在 Python 语言中，非“0”的数值或者字符串被视为“True” |
| not 2 | False | |
| not 3>2 | False | |
| not 'a' | False | |
| not 3<2 | True | |
| 2 and 3 | 3 | |
| 3 and 2 | 2 | |
| 'a' and 'b' | 'b' | |

续表

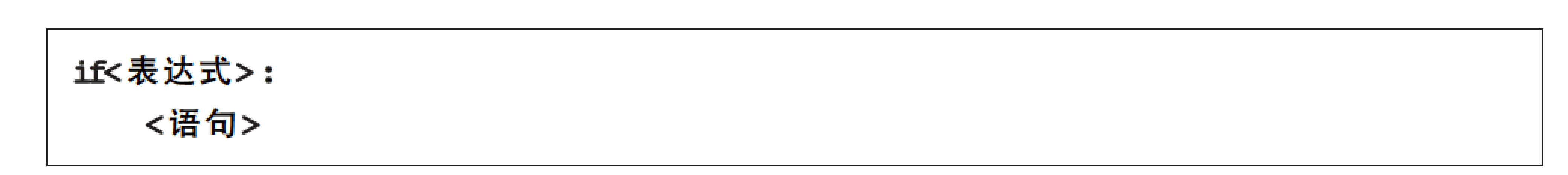
| 示例 | | 示例说明 |
|---|---|---|
| 'b' and 'a' | 'a' | 在使用逻辑运算符时，最需要注意的一点是短路逻辑（或叫做惰性求值），它表达的意思是：逻辑运算是自左向右进行的，如果左边已经决定结果，就不会再去做右边的计算。此外，在 Python 语言中，非“0”的数值或者字符串被视为“True” |
| 2>1 and 3>2 | True | |
| 2>1 and 3<2 | False | |
| 2<1 and 3<2 | False | |
| 2 or 3 | 2 | |
| 3 or 2 | 3 | |
| 'a' or 'b' | 'a' | |
| 'b' or 'a' | 'b' | |
| 2>1 or 3>2 | True | |
| 2>1 or 3<2 | True | |
| 2<1 or 3<2 | False | |

# 4.3 分支语句

Python 语言通过 if、elif 和 else 语句实现单分支、二分支和多分支结构。接下来对以上三种结构进行详细的阐述。

## 4.3.1 单分支结构：if 语句

if 语句的语法格式如下：



```
if<表达式>:
    <语句>
```

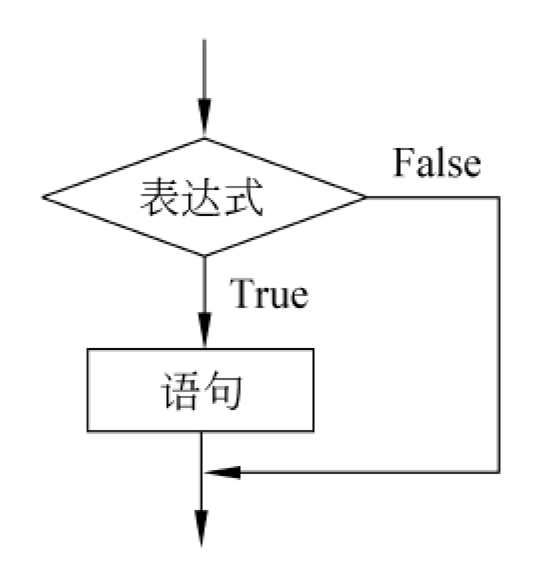



图 4-15 if 语句（单分支结构）

这种 if 语句的执行过程如图 4-15 所示。其中 if 后面的表达式为一个或者多个语句序列，语句块中的语句通过与 if 所在行形成缩进表达包含关系。if 语句首先根据表达式给出判断结果“True”或“False”；如果为“True”，则执行相应语句块中的语句序列，然后控制转向下一条语句；如果结果为“False”，语句块中的语句会被跳过。

**【例 4-7】** 学生成绩分类。一门课的成绩（score）实行五级计分制，见表 4-2，通过 Python 单分支 if 语句将学生具体分数转化为五级计分制。

**表 4-2　五级计分制标准**

| 分数(score) | 五级计分结果 | 分数(score) | 五级计分结果 |
| --- | --- | --- | --- |
| score≥90 | A | 70>score≥60 | D |
| 90>score≥80 | B | score<60 | F |
| 80>score≥70 | C | | |

通过 Python 的单分支 if 语句设计的程序如下。

```
def score(x):
    if x>=90:
        return 'A'
    if 80<=x<90:
        return 'B'
    if 70<=x<80:
        return 'C'
    if 60<=x<70:
        return 'D'
    if x<60:
        return 'F'
print(score(90))
```

运行结果:

```
'A'
```

**【例 4-8】** 输入三个数 x、y 和 z,由大到小顺序输出。与例 4-7 相比,这里稍微复杂一些,先设计其伪代码算法如下。

如果 x<y,将 x 与 y 对换(x 是 x 与 y 中的大者)。

如果 x<z,将 x 与 z 对换(x 是 x 与 z 中的大者;因此 x 是三个数中最大者)。

如果 y<z,将 y 与 z 对换(y 是 y 与 z 中的大者;因此 y 是三个数中第二大者)。

然后将 x、y、z 顺序输出即可。

```
def sort(x,y,z):
    if x<y:
        x,y=y,x
    if x<z:
        x,z=z,x
    if y<z:
        y,z=z,y
    return x,y,z
print(sort(3,7,1))
```

运行结果：

```
(7,3,1)
```

### 4.3.2 二分支结构：if-else 语句

if-else 语句形成二分支结构，语法格式如下：

```
if<表达式>:
    <语句 1>
else:
    <语句 2>
```

if-else 语句的执行过程如图 4-16。语句 1 是 if 条件满足后执行的一个或多个语句序列，语句 2 是 if 条件不满足后执行的语句序列。二分支结构用于区分表达式的两种可能，即“True”和“False”。

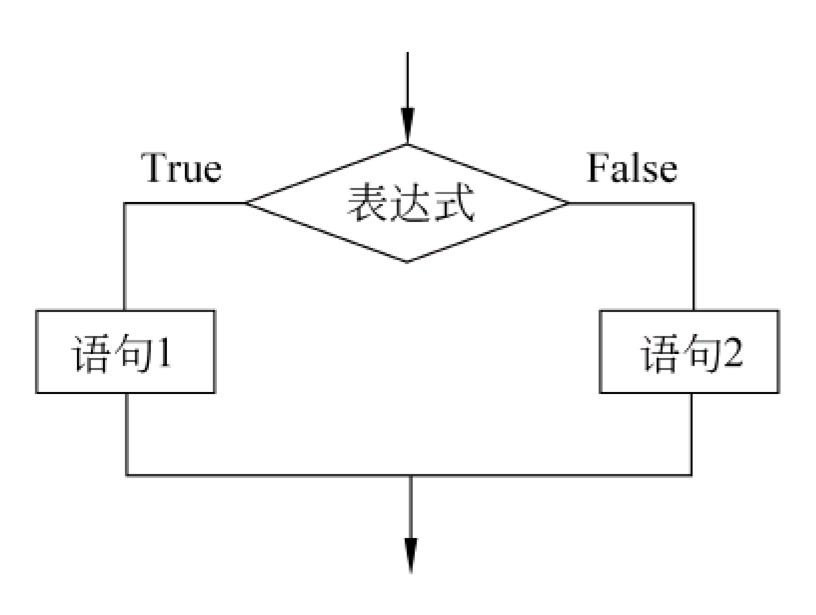


图 4-16 if-else 语句(二分支结构)

**【例 4-9】** 考取驾照已成为当今人们的一个基本社会技能，在满足报名的条件下，首先需通过科目 1 的考试，科目 1 的成绩分为“通过”(成绩大于等于 90 分，满分 100)和“不通过”(成绩小于 90 分)两种情况。基于 Python 编写程序如下：

```
def score(s):
    if s< 90:
        return '不通过'
    else:
        return '通过'
print(score(89))
```

运行结果：

```
不通过
```

if-else 语句还有一种更加简洁的方式，适合通过判定返回特定的值，其语法格式如下：

```
<表达式 1> if <条件> else <表达式 2>
```

因此，例 4-9 的程序可修改为如下：

```
s=eval(input('输入科目 1考试成绩：'))
print('通过' if s>=90 else '不通过')
```

运行结果：

```
输入科目 1考试成绩：89
不通过
```

### 4.3.3 多分支结构：if-elif-else 语句

if-elif-else 语句描述多分支结构，如图 4-17 所示。其语法格式如下：

```
if <表达式 1>:
    <语句 1>
elif <表达式 2>:
    <语句 2>
    …
else:
    <语句 n>
```

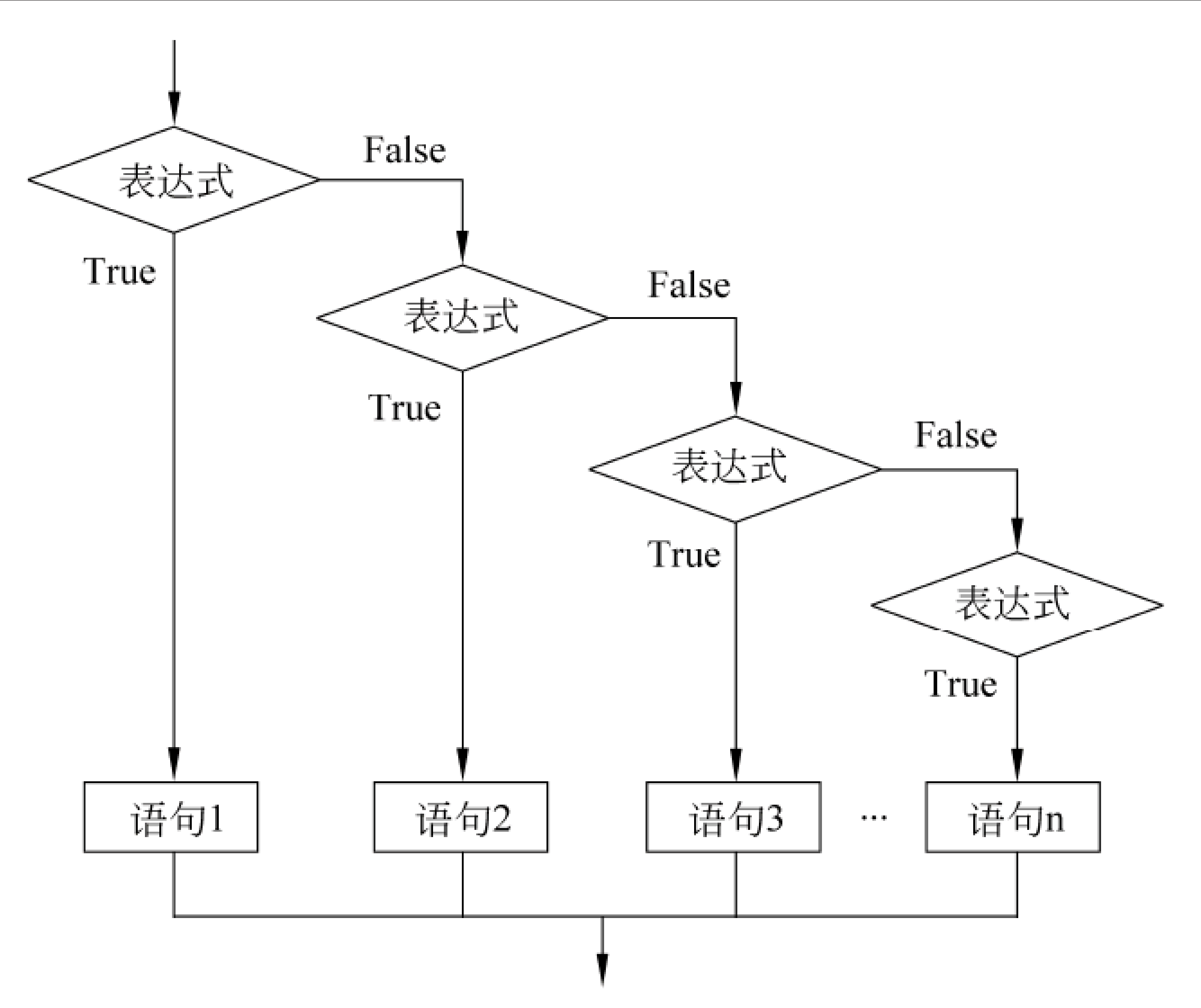



图 4-17 if-elif-else 语句(多分支结构)

多分支结构是二分支结构的扩展，这种形式通常应用于同一个判断条件的多条执行路径。Python 语句依次寻找第一个为"True"的表达式，执行该表达式后的语句块，结束后跳出整个 if-elif-else 结构，继续执行后面的语句。

**【例 4-10】** 用 if-elif-else 语句实现例 4-7 的 Python 程序如下。

```
def score(x):
    if x>=90:
        return 'A'
    elif 80<=x<90:
        return 'B'
    elif 70<=x<80:
        return 'C'
    elif 60<=x<70:
        return 'D'
    else:
        return 'F'
print(score(90))
```

运行结果：

```
'A'
```

# 4.4 循环控制语句

在各类程序语言中，根据循环执行次数，可分为确定次数循环和非确定次数循环。确定次数循环指的是循环体对循环次数有明确的定义，例如遍历一个字符串，这类循环在Python中称为遍历循环，其中循环次数采用遍历结构中的元素个数来体现，具体采用for语句实现，for循环也同样适用于其他确定次数的循环体。而非确定次数循环表示程序不确定循环体可能的执行次数，而通过条件判断是否继续执行循环体，Python定义了while语句来实现非确定次数的循环。需要注意的是，while语句同样可应用于确定次数的循环，参考例4-3。

## 4.4.1 for语句

Python通过关键字for实现遍历循环，其语法格式如下：

```
for <循环变量> in <遍历结构>:
    <语句>
```

for循环的“遍历结构”中的元素是确定的，因此循环次数也是明确的。遍历循环就是基于遍历结构中的每一个元素执行一次语句，其中遍历结构可以是range()函数、字符串、列表、文件和组合数据类型等，常用的集中遍历方式如下。

### 1. 循环n次

range(n)函数将产生n个连续整数，范围为0，1，2，…，n－1。循环变量将逐个访问

序列中的每个整数值。例如，range(10)将产生 10 个整数，即 0，1，2，3，4，5，6，7，8，9。如果通过 range()函数产生的数值生成列表，即使用 list(range(n))；如果通过 range()函数产生的数值生成元组，即使用 tuple(range(n))。

**【例 4-11】** 计算 $\sum_{i=1}^{100} i$。

```
s=0
for i in range(101):
    s=s+i
print("从 1 累加到 100 的和：",s)
s=0
for i in list(range(101)):
    s=s+i
print("从 1 累加到 100 的和：",s)
s=0
for i in tuple(range(101)):
    s=s+i
print("从 1 累加到 100 的和：",s)
```

运行结果：

```
从 1 累加到 100 的和：5050
从 1 累加到 100 的和：5050
从 1 累加到 100 的和：5050
```

### 2. 遍历字符串

```
for <循环变量> in S:                #S表示一个字符串
    <语句>
```

**【例 4-12】** 输出字符串"BJTU"的每一个字母，其 Python 程序代码如下：

```
s='BJTU'
for i in s:
    print(i)
```

运行结果：

```
B
J
T
U
```

### 3. 遍历列表

```
for <循环变量> in ls:                #ls表示一个列表
    <语句>
```

【例 4-13】 输出列表 ls=['bjtu','sjtu','xjtu']的每一个元素,其 Python 程序代码如下:

```
ls=['bjtu','sjtu','xjtu']
for i in ls:
    print(i)
```

运行结果:

```
bjtu
sjtu
xjtu
```

### 4. 遍历元组

```
for <循环变量> in t:                #t表示一个元组
    <语句>
```

【例 4-14】 求元组(6,7,8,9,10)中所有元素的和,其 Python 程序代码如下:

```
t=(6,7,8,9,10)
s=0
for i in t:
    s=s+i
print(s)
```

运行结果:

```
40
```

### 5. 遍历文件的每一行

```
for <循环变量> in fi:                #fi表示一个文件
    <语句>
```

for 语句除去上述的遍历结构,还有一类扩展模式,其语法格式如下:

```
for <循环变量> in <遍历结构>:
    <语句 1>
else:
    <语句 2>
```

**注意**：在这种扩展模式中，当 for 语句循环正常执行结束后，会继续执行 else 语句中的内容。else 语句只在循环正常结束后才能执行。

**【例 4-15】** 遍历"bjtu"字符串。基于 for-else 语句的 Python 程序代码如下：

```
s= 'bjtu'
for i in s:
    print("循环进行：",i)
else:
    print("循环结束。")
```

运行结果：

```
循环进行：b
循环进行：j
循环进行：t
循环进行：u
循环结束。
```

## 4.4.2 while 语句

while 语句是 Python 中另外一个循环语句，其更多应用于条件循环，也称为非确定次数循环，循环直到循环条件不满足才结束，不需要提前确定循环次数。其基本语法格式如下：

```
while <表达式>:
    <语句>
```

其表达式与 if 语句的判断条件相同，结果为"True"或"False"。while 语句语义简单，当表达式的结果一直为"True"时，循环体重复执行语句块中的语句；当条件为"False"时，循环终止，继续执行与 while 同级别缩进的后续语句。

**【例 4-16】** 计算 $\sum_{i=1} \frac{1}{i}$，直到 $\frac{1}{i^3} < 10^{-4}$ 结束循环，其 Python 程序代码如下：

```
s=0
i=1
```

```
while 1/i**3>=10e-4:
    s=s+1/i
    i=i+1
print(s)
```

运行结果：

```
2.9289682539682538
```

类似于 for-else 语句，无限循环同样保留了 while-else 的扩展语句，其语法格式如下：

```
while  <表达式>:
    <语句 1>
else:
    <语句 2>
```

while-else 语句表示当 while 循环正常结束后，程序将继续执行 else 语句中的内容。需要注意的是 else 语句只在循环正常执行结束后才执行。因此可以在“语句 2”中放置判断循环结束的语句。

**【例 4-17】** 遍历"bjtu"字符串。基于 while-else 语句的 Python 程序代码如下：

```
s,n='BJTU',0
while n<len(s):
    print("循环执行：",s[n])
    n=n+1
else:
    print("循环结束。")
```

运行结果：

```
循环执行：B
循环执行：J
循环执行：T
循环执行：U
循环结束。
```

从 for 语句和 while 语句的定义及相关例子可以看出，for 语句一般应用于确定性循环，而 while 语句多应用于非确定性循环。但是在循环中，某些时候需要终止循环或者跳出本次循环，Python 为了实现这一功能，定义了循环关键字 break 和 continue。

### 4.4.3 break 和 continue

Python 语言中的 break 和 continue 类似于 C 语言，break 语句用于跳出 for 或者

while 循环；continue 语句用于结束当前当次循环，在 for 语句中，循环程序流程接着遍历，在 while 语句中，则继续求解循环条件。接下来用例 4-18 来观察两者的不同。

**【例 4-18】** Python 环境下，for 循环语句对 break 语句和 continue 语句的使用，Python 程序代码如下：

```
#break语句示例
s='BJTU'
for i in s:
    if i=='J':
        break
print(i)
```

运行结果：

```
B
```

```
#continue语句示例
s='BJTU'
fori in s:
    if i=='J':
        continue
print(i)
```

运行结果：

```
B
T
U
```

**【例 4-19】** 把 100～120 的不能被 3 整除的数输出，Python 程序代码如下：

```
#continue语句示例
s='BJTU'
fori in s:
    if i=='J':
        continue
print(i)
```

运行结果：

```
100  101  103  104  106  107  109  110  112  113  115  116  118  119
```

当 i 能被 3 整除时，执行 continue 语句，结束本次循环（也就是跳过 print 函数语句），只有 i 不能被 3 整除时才执行 print 函数。例 4-19 也可通过如下程序实现：

```
for i in range(100,120):
    if i%3!=0:
        print('{:<5}'.format(i),end='')
```

运行结果：

```
100  101  103  104  106  107  109  110  112  113  115  116  118  119
```

在例 4-19 中使用 continue 语句，主要是为了进一步阐述该关键字的作用。此外，从例 4-19 可以看出，break 和 continue 的区别是，continue 只结束本次循环，但不会终止整个循环结构，而 break 则终止了整个循环。

## 4.4.4 程序的异常处理语句

对于程序设计者而言，尤其是初学者，编写程序过程中会出现一些错误或异常，导致程序终止。捕捉程序异常可使用 try-except 语句。try-except 语句用来检测 try 语句块中的错误，从而让 except 语句捕获异常信息并处理；如果你不想在异常发生时结束程序，只需在 try 里捕获它即可。

异常处理语句的语法格式如下：

```
try:
    <语句 1>                #运行别的代码
except<名字>:
    <语句 2>                #如果在 try 部分引发了'name'异常
except<名字>,<数据>:
    <语句 3>                #如果引发了'name'异常,获得附加的数据
else:
    <语句 4>                #如果没有异常发生
```

try 的工作原理是，当开始一个 try 语句后，Python 就在当前程序的上下文中作标记，这样当异常出现时就可以回到这里，try 子句先执行，接下来会发生什么取决于执行时是否出现异常。

如果当 try 后的语句执行时发生异常，Python 就跳回到 try 并执行第一个匹配该异常的 except 子句，异常处理完毕，控制流就通过整个 try 语句（除非在处理异常时又引发新的异常）。

如果在 try 后的语句里发生了异常，却没有匹配的 except 子句，异常将被递交给上层的 try，或者到程序的最上层（这样将结束程序，并打印缺省的出错信息）。

如果在 try 子句执行时没有发生异常，Python 将执行 else 语句后的语句（如果有 else 的话），然后控制流通过整个 try 语句。

**【例 4-20】** 除数为 0 的运算。Python 程序代码如下：

```
x=2
y=0
z=x/y
print(z)
```

运行结果：

```
ZeroDivisionError: division by zero
```

运行结果是：出现“ZeroDivisionError: division by zero”异常。程序因为ZeroDivisionError而中断了，语句print没有运行。为了处理异常，我们使用try-except，更改代码如下：

```
x=2
y=0
try:
    z=x/y
    print(z)
except ZeroDivisionError:
    print('error')
```

运行结果：

```
error
```

除try-except语句外，Python语言还定义了try-finally语句，其语法格式为：

```
try:
    <语句 1>
finally:
    <语句 2>            #退出 try时总会执行
```

需要注意的是，可以使用except语句或者finally语句，但两者不能同时使用。else语句也不能与finally语句同时使用。

**【例 4-21】** 打开一个文件，没有可写权限，其Python程序代码如下：

```
try:
    fh=open("testfile", "w")
    fh.write("This is my test file for exception handling!!")
finally:
    print ("Error: can\'t find file or read data")
```

运行结果：

```
Error: can't find file or read data
```

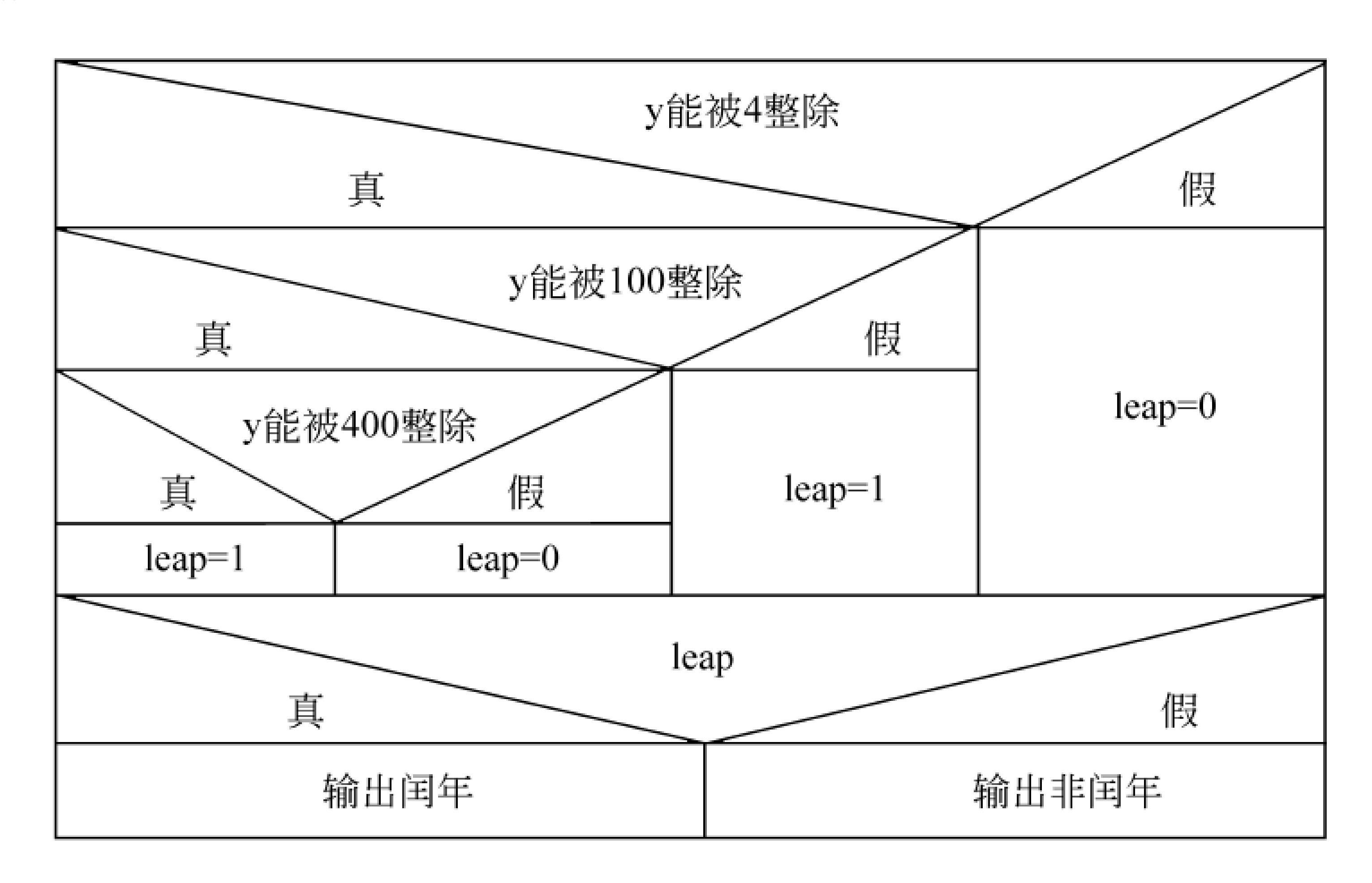
## 4.5　控制结构综合案例

**【例 4-22】** 判断某一年是否是闰年。闰年的条件如下：

① 能被 4 整除，但是不能被 100 整除的年份都是闰年，例如 2004、2008。

② 能被 400 整除的年份是闰年，例如 1600 和 2000。

针对闰年的条件，设计如下算法，其中 y 表示年份，leap 表示是否闰年的信息，若 leap=1 表示该年份为闰年，若 leap=0 则表示该年份为非闰年，最后判断 leap 是否为 1 来输出是否为闰年信息。




```
def leapyear(y):
    if y%4==0:
        if y%100==0:
            if y%400==0:
                leap=1
            else:
                leap=0
        else:
            leap=1
    else:
        leap=0
    if leap==1:
        print("{}是闰年".format(y))
```

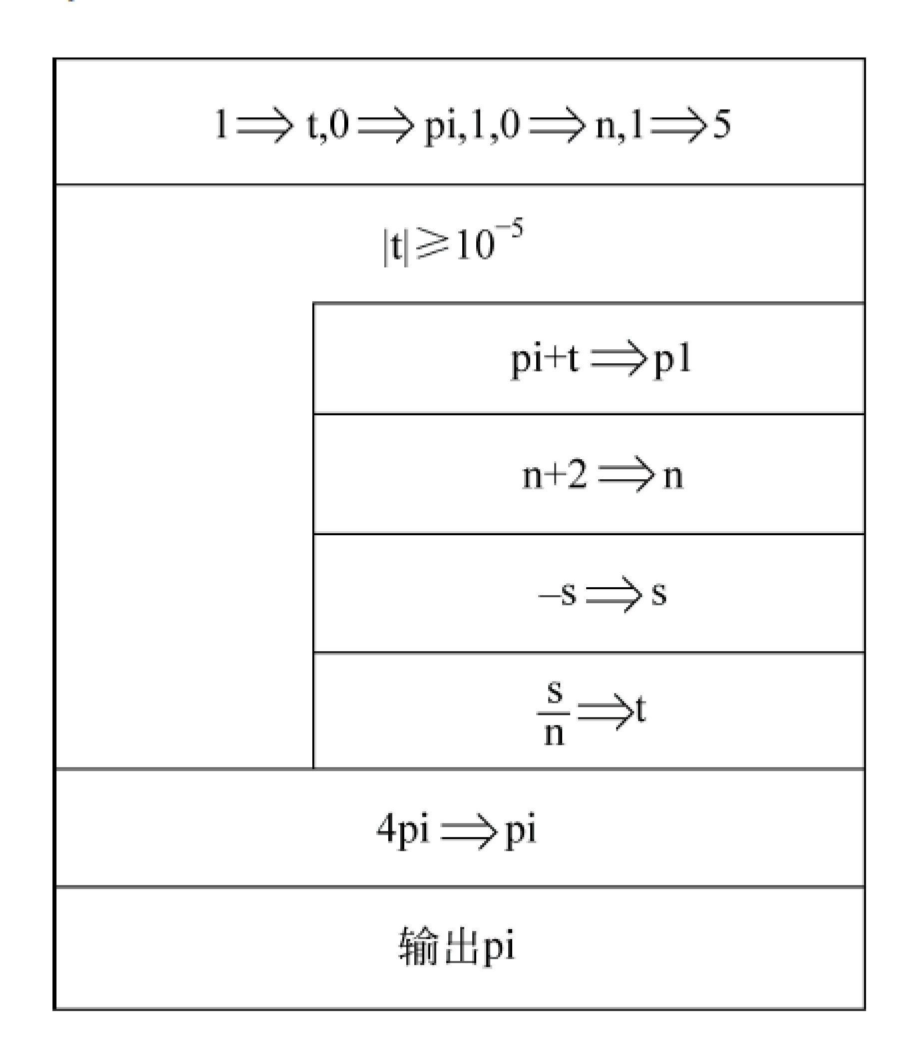
```
    else:
        print("{}不是闰年".format(y))
print(leapyear(2000),leapyear(2004),leapyear(2017))
```

运行结果：

```
2000是闰年,
2004是闰年,
2017不是闰年
```

**【例 4-23】** 利用$\frac{\pi}{4}\approx 1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots$求 $\pi$ 的近似值，直到最后一项的绝对值小于 $10^{-5}$ 为止。

基于该表达式实现的 Python 程序如下：




```
t,pi,n,s=1,0,1.0,1
while abs(t)>1e-5:
    pi=pi+t
    n=n+2
    s=-s
    t=s/n
pi=pi*4
print("pi的值",pi)
```

运行结果：

```
pi的值 3.1415726535897814
```

**【例 4-24】** 编程统计某一字符串中各字母出现的次数(不区分大小写)。

该问题的算法如下：创建一个字典，用字母作为键，该字母出现的次数作为对应的键值。某个字母第一次出现，相应地在字典里面加一个键(及其对应的键值)。此后再遇到这个字母，就在其键值上累加即可。基于该算法的 Python 程序如下：

```
def histogram(input_str):                    #input_str表示输入的字符串
    b=input_str.upper()                      #将输入字符串全部大写
    d=dict()                                 #创建字母,统计字母出现次数
    for c in b:
        if ord(c)>=65 and ord(c)<=90:        #判断该字符是否在字母表中
            if c not in d:
                d[c]=1
            else:
                d[c]+=1
    return d
print histogram('Nice to meet you!')
```

运行结果：

```
{'N': 1, 'I': 1, 'C': 1, 'E': 3, 'T': 2, 'O': 2, 'M': 1, 'Y': 1, 'U': 1}
```

**【例 4-25】** 一个班级有若干名同学，且假定同学的名字是唯一的。某学期大家学习 Python 程序设计课程，期末成绩分为 A(卷面成绩大于或等于 90)、B(卷面成绩大于或等于 60 且小于 90)和 C(卷面成绩小于 60)。现在知道同学们的卷面成绩，要求统计各个成绩段的同学名单。

可将上述统计问题分解为两部分：①将学生的卷面成绩转化为等级成绩；①在此基础上，进行成绩统计。其 Python 程序如下：

```
def score(d):  #d is a dictionary in Python
    new_score=dict()
    for key in d:
        if d[key]>=90:
            new_score[key]='A'
        if d[key]<60:
            new_score[key]='C'
        if d[key]>=60 and d[key]<90:
            new_score[key]='B'
    return new_score
def inverse(d):
    inv=dict()
    for key in score(d):
        val=score(d)[key]
        if val not in inv:
            inv[val]=[key]
```

```
        else:
            inv[val].append(key)
    return inv
A={'Zhang':60,'Yang':65,'Tom':49,'Hou':95}
Print inverse(A)
```

运行结果：

```
{'B': ['Zhang', 'Yang'], 'C': ['Tom'], 'A': ['Hou']}
```

# 本章小结

(1) 理解程序的三种基本控制结构：顺序结构、选择结构和循环结构；
(2) 掌握关系运算符和关系表达式；
(3) 掌握逻辑运算符和逻辑表达式；
(4) 掌握并熟练应用 if 语句；
(5) 掌握并熟练应用循环控制语句；
(6) 基本掌握并应用本章知识设计较复杂的 Python 程序。

# 习　　题

## 1. 简答题

(1) 什么是算法？请从日常生活、学习和工作中找出两个例子，描述它们对应的算法。

(2) 阐述程序设计的三种基本结构的特点；在此基础上，设计两种基本结构(基于基本结构的特点)。

(3) 解释算术运算、关系运算和逻辑运算。

(4) Python 语言中如何表示"真"和"假"？

(5) 传统流程图和 N-S 流程图各有何特点？其各自优势体现在哪里？

## 2. 计算题

(1) 设 x=4,y=5,z=6,计算 x+y>z and x==z。
(2) 设 x=4,y=5,z=6,计算 x or y or z。
(3) 设 x=4,y=5,z=6,计算 x and y and z。
(4) 设 x=4,y=5,z=6,计算 not x>y and not (x>z)。
(5) 设 x=4,y=5,z=6,计算(y>x) and (z>x) and 0。

## 3. 程序设计题

(1) 有 3 个杯子(记为 A、B 和 C),其中 A 盛放的是苹果汁、B 盛放的是西瓜汁、C 是空杯,要求 A 和 B 互换。

(2) 依次输入 5 个数,将其升序排列。

(3) 判断某一个数是否是完数(完数就是其所有真因子的和,恰好等于它本身)。

(4) 求两个正整数的最大公因子。

(5) 统计某一字符串中出现 2 次的字母。

(6) 2017 年某银行的定期存款利率表如下:

| 类型 | 年利率/% | 类型 | 年利率/% |
|---|---|---|---|
| 三个月 | 1.1 | 两年 | 2.1 |
| 半年 | 1.3 | 三年 | 2.3 |
| 一年 | 1.5 | 五年及以上 | 2.8 |

现在存入 100 000 元,五年后本金加利息各是多少?

(7) 给定一个 n(n>2)阶实数方阵,确定矩阵元素中的最大值。

(8) 在 1849 年,阿尔方·德·波利尼亚克提出了一般的猜想:对所有自然数 k,存在无穷多个素数对(p,p+2k)。k=1 的情况就是孪生素数猜想。孪生素数猜想是一大数学难题,在近一个世纪以来,一大批数学家在努力解决这一伟大的猜想。请编程计算 100 到 10000 内有多少孪生素数对。

(9) 假如一个班级有 30 名学生,这当中两个人同一天出生的概率是多大?(提示:用 randint 函数来生成随机的生日,这个函数包含在 random 模块中)

(10) 用牛顿法求解下面方程在 1.5 附近的根。

$$2x^3 - 4x^2 + 3x - 6 = 0$$

(11) 通过至少 3 种不同的排序算法实现对 6、3、7、8、5 降序排列(排序算法可自行构造或者通过互联网搜索)。

(12) 当输入为 5 时,写出下面程序运行结果。

```
import math
def fun(num):
    for j in range(2,math.floor(num/2)+1):
        if num%j==0:
            return False
        else:
            return True
def main():
    n=eval(input('Please input an integer:'))
    c=0
    for i in range(2,n+1):
        if fun(i):
            c+=1
    print(c)
main()
```

(13) 写出下面程序的运行结果。

```
x='BJTU'
for i in x:
    for j in range(1,3):
        if i=='J':
            continue
        else:
            print(i,end='')
```

(14) 打印输出一个 10×10 的乘法表。

(15) 编写程序输出以下图形。

```
         *
        ***
       *****
      *******
     *********
    ***********
   *************
  ***************
 *****************
*******************
```

# 第 5 章　函数设计初步

## 5.1　函数定义

### 5.1.1　程序设计函数的起源

数学中的函数 y=f(x)可以实现某种数据运算功能，例如 y=sin(x)用来计算自变量 x 的正弦值 y。程序设计中也有函数的概念。程序中的函数是可以实现某个特定功能的小程序块(block)。每当程序需要实现该特定功能时，只需调用事先写好的函数，不必每次重复编写相同功能的代码。当需要改变函数功能时，只需要修改函数中的代码，则程序中所有调用该函数的地方都会同步修改。因此使用函数可以实现代码复用，提高编程效率。

使用函数可以简化程序结构。复杂的编程问题可以分解成若干简单的子问题，每个子问题用函数来解决。函数像积木块一样通过灵活组合构建起复杂程序。这种简化问题的方法体现了“自顶向下、模块化编程”的程序设计思想。

使用函数可以使程序更容易阅读。函数使程序结构层次分明，好的函数名可以直接体现程序的功能，易于阅读和理解。

使用函数还可以让程序更容易调试和测试。如果程序由一系列的函数组成，程序员可以编写程序调用每个函数，在各种测试条件下测试函数的运行状态。当所有函数都能正常运行时，程序的运行就基本没有问题。当程序运行出错时，也很容易定位到某个函数并找出错误。

### 5.1.2　函数的定义

Python 包含内建函数和用户自定义函数。内建函数是 Python 事先定义好的程序，用户可以直接使用，如 pow()、input()、print()等。用户也可以自定义函数，方法如下：

```
def 函数名 (参数 1,参数 2…):
    函数体
    return 返回值
```

其中，第一行是 def 语句，用关键字 def(def 是英文 define 的缩写)定义函数。def 与

函数名之间有一个空格。函数名通常体现函数功能，且必须符合变量命名的规则。函数名后面圆括号中的参数1、参数2等称为“形式参数”，简称为形参。它的作用是实现调用程序与被调用函数之间的联系。通常把函数要处理的数据、影响函数功能的参数或者函数的运行结果作为形参。形参的个数可以是零个、一个或多个，当参数个数为零时，圆括号也要保留。def语句以冒号结尾，表示后面的函数体与def语句之间有缩进关系（通常是4个空格）。函数体是函数每次被调用时执行的代码。return语句是可选项，它可以在函数体内任何地方出现，表示函数执行到此结束，控制权返回给调用程序，同时返回处理结果。

在图5-1的程序中，第1行定义了一个名称为Sum的函数，形参为a、b、c。第2、3行是函数体，其功能是计算参数a、b、c的和，return语句返回求和结果。第4行print语句与第1行def语句没有缩进关系，因此不属于Sum函数范围，它的功能是调用Sum函数计算10、2、3的和并显示计算结果。在调用函数时，必须给函数传递具体的参数值，此时参数称为“实际参数”，简称实参。10、2、3就是实参。实参可以是常量、变量、表达式、函数等，无论何种类型，在进行函数调用时都必须具有确定的值，才能让函数运行。

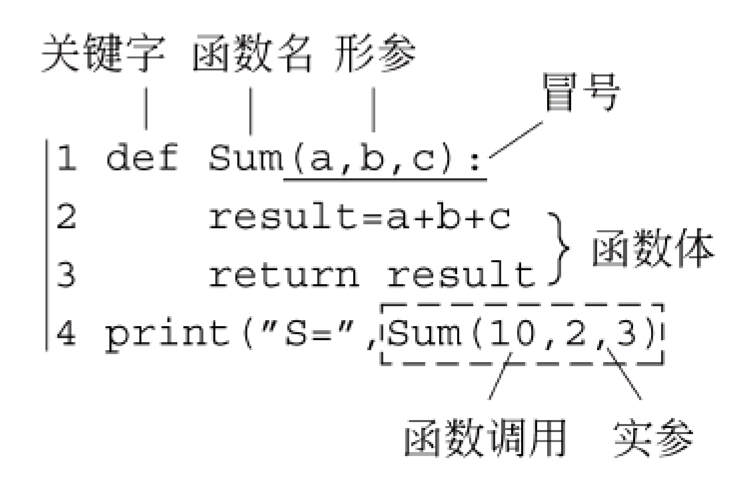


图5-1 函数定义及调用示例

该程序的运行结果如下：

```
======RESTART: C:/Python36-32/图 5.1 程序 .py======
S=15
```

如果将程序中的第4行放到第1行，结果如下：

```
======RESTART: C:/Python36-32/图 5.1 程序 .py======
Traceback (most recent call last):
  File "C:/Python36-32/图 5.1 程序 .py", line 1, in<module>
    print("S=",Sum(10,2,3))
NameError: name 'Sum' is not definedPython
```

Python提示名字为“Sum”的函数未进行定义。这是因为在调用Sum函数前并没有定义该函数。因此调用函数时必须确保该函数之前已经定义。

再看两个函数定义的例子。

**【例5-1】** 定义一个函数，将华氏温度转换为摄氏温度。

```
def fTemp (t):
    converTemp= (5/9) * (t-32)
    return converTemp
```

函数的参数是华氏温度，返回其对应的摄氏温度，例如：

```
>>> fahTemp=eval(input("请输入华氏温度："))
70
>>> celTemp=int(fTemp(fahTemp))
>>> print("转换后的摄氏温度是：", celTemp, "度")
转换后的摄氏温度是：21 度
```

代码中的 int()函数将摄氏温度浮点数转换为整数。

**【例 5-2】** 定义一个函数，提取中文姓名中的姓氏和名字。假设姓氏与名字之间有一个空格。

```
def getName(name):
    nameList=name.split()
    giveName=nameList[0]
    firstName=nameList[-1]
return giveName,firstName
```

该函数的参数是一个姓名字符串，姓氏与名字之间有一个空格。函数中 name.split()的功能是将字符串 name 按照空格分成不同元素保存到一个列表中，在本例中，列表有两个元素：姓氏和名字。函数返回姓氏和名字两个结果，在调用函数时应定义两个变量分别保存它们：

```
>>> Name=input("请输入姓名,姓氏和名字之间以一个空格分开：")
请输入姓名,姓氏和名字之间以一个空格分开：诸葛亮
>>> givename, firstname=getName(Name)
>>> print("姓氏：", givename, "名字：", firstname)
姓氏：诸葛 名字：亮
```

## 5.1.3 匿名函数

函数定义中有一类特殊函数，称为匿名函数或 lambda 函数。它使用 lambda 关键字，一般形式如下：

```
f=lambda 参数 1, 参数 2, …：表达式
```

这种函数省略了 def 声明函数的标准步骤，因此被称为匿名函数。其本质是一个表达式，能接收任何数量的参数并返回表达式的值。例如：

```
>>> f=lambda x, y, z: x+y+z
>>> f (1,2,3)
6
>>> g=lambda x: lambda y: x+y
```

```
>>>a=g(4)
>>>a(10)
14
>>>g(4)(10)
14
```

代码中 g 返回的是一个 lambda 函数 lambda y：x+y。当 x=4 时，a=g(4)=lambda y:4+y。当 y=10 时，a(10)=4+10=14。两个表达式也可以简化为 g(4)(10)。

# 5.2 函数的参数传递

函数有三种方法将实参传递给形参，即按照位置传递参数、按照关键字传递参数、按照默认值传递参数。

## 5.2.1 按照位置传递参数

函数调用时默认采用该方法传递参数，其形式如下：

```
函数名 (参数 1,参数 2,…)
```

这种参数传递方法要求形参和实参的个数必须一致，并且一一对应，即相同位置的实参向相同位置的形参传递参数。当实参是一个表达式时，先要计算表达式的值，再将它传递给形参。例如图 5-1 的程序中 print 函数在调用 Sum 函数时就是按照位置传递参数：

```
Sum (10,2,3)
```

这里 a=10，b=2，c=3。

## 5.2.2 按照关键字传递参数

当函数需要传递很多参数时，按照位置传递参数很容易出错，此时可以按照关键字传递实参，其形式如下：

```
函数名 (参数名 1=值 1,参数名 2=值 2,…)
```

这种参数传递方法在实参前添加了参数名，即关键字。关键字明确了每个参数的含义，这样即便参数顺序被打乱，参数的位置发生改变，也不会影响参数的传递。例如图 5-1 的程序如果需要对 10 个值求和，可以采用关键字传递实参。同时，关键字参数可以在函数中提供默认值。

```
Sum (a=10, b=2, c=3, d=…)
```

### 5.2.3 按照默认值传递参数

在定义函数时可以给形参指定默认值。在调用函数时如果形参提供了数值，则使用给定的参数值，如果没有提供数值，则会使用默认值。例如图 5-1 中的程序，如果函数定义如下：

```
def Sum (a,b,c=2)
    result=a+b+c
    return result
```

此时 Sum 函数有三个形参 a、b 和 c，其中 c 指定了默认值 3。如果调用函数如下：

```
>>>print ("S=", Sum (10,2,3))
S=15
```

结果为 15。因为调用函数时形参 c 提供了参数 3，c 等于 3，而不是默认值 2，求和结果是 10＋2＋3＝15。如果调用函数如下：

```
>>>print("S=", Sum(10,2))
S=14
```

结果为 14。因为此时实参 10、2 按照位置顺序分别传递给形参 a、b，而 c 没有提供实参，因此其值等于默认值 2，求和结果是 10＋2＋2＝14。要注意的是，如果函数定义时给某个形参指定了默认值，则该参数必须定义在无默认值的形参后面，因此 c 要定义在 a、b 之后。

### 5.2.4 可变数量的参数传递

某些情况下，在定义函数时无法确定参数个数，Python 允许函数设计可变数量的参数。此时函数定义形式如下：

```
函数名 (* 参数)
```

在参数名前面加一个“ * ”，表示参数是以形参名为标识符的元组，元组中的元素个数可以是零个、一个或多个。例如，定义函数：

```
def f(* a):
    print(a)
```

调用这个函数：

```
>>>f("Hello")
('Hello',)
```

结果表明参数 a 是一个以字符串“Hello”为元素的元组。又如下例：

```
>>>f(1,2,3)
(1, 2, 3)
```

返回结果是一个包含三个参数 1、2、3 的元组。如果不提供任何参数：

```
>>>f ()
()
```

则返回一个空元组。如果按照关键字传递参数，例如：

```
>>>f(a=1)
Traceback (most recent call last):
    File "<pyshell #257>", line 1, in<module>
        f(a=1)
TypeError: f() got an unexpected keyword argument 'a'
```

结果提示出错。此时应采用如下方式定义函数：

```
函数名 (**参数)
```

在参数名前加“**”，表示参数的数据类型是字典，其中关键字(参数名)为“键”，参数值为“值”。例如：

```
def f(**a):
    print(a)
```

此时再按关键字传递参数：

```
>>>f (x=1, y=2, z=3)
{'x': 1, 'y': 2, 'c': 3}
```

返回结果是一个字典参数，其中每个元素以关键字(参数名)为键、参数值为值。

综上所述，按照位置和按照关键字传递参数时，参数数量是固定不变的。如果参数数量不定或可变，需要在参数名前加“*”或“**”。加“*”表示参数是元组参数，加“**”表示参数是字典参数(键值对)。在定义函数时，可以混合使用多种参数传递方式，此时要遵循以下规则：

（1）关键字参数应放在位置参数后面。

（2）元组参数必须在关键字参数后面。

（3）字典参数要放在元组参数后面。

在调用函数时，首先按位置顺序传递参数，其次按关键字传递参数。多余的非关键字参数传递给元组，多余的关键字参数传递给字典。

**【例 5-3】** 下列函数采用字典参数，观察函数调用结果。

```
def story(**info):
    print('人生苦短,{name}爱{language}'.format(**info))
```

调用函数，结果如下：

```
>>>story(name='Gudio', language='Python')
人生苦短,Gudio爱 Python
>>>story(name='James', language='Java')
人生苦短,James爱 Java
>>>params={'name':'Dennis','language':'C'}
>>>story(**params))
人生苦短,Dennis爱 C
>>>del params['name']
>>>story(name='stroke of genius',**params)
人生苦短,stroke of genius爱 C
```

**【例 5-4】** 函数定义如下：

```
def power (a, b, *c):
    if c:
        print('接收多余的参数：',c)
    return a**b
```

调用函数，进一步理解元组参数的传递方法：

```
>>>power(2, 3)
8
>>>power(3, 2)
9
>>>power(b=3, a=2)
8
>>>arg1=(5, )*2
>>>power(*arg1)
3125
>>>arg2=(5,6)*2
>>>power(*arg2)
接收多余的参数：(5, 6)
```

```
15625
>>>power(3,3,'Hello,world')
接收多余的参数: ('Hello, world',)
27
```

函数 power 定义了三个形参,其中 a 和 b 是普通参数,c 是元组参数。if c 语句判断是否存在参数 c,存在的话给出提示并计算 a**b,否则直接计算 a**b。代码中 arg1=(5,)*2=(5,5),是一个元组。元组中的元素按照位置顺序赋值给形参,因此 power(*arg)等价于 power(5,5),没有参数 c,直接计算 5**5,返回 3125。arg2=(5,6)*2=(5,6,5,6),power(*arg2)等价于 power(5,6,5,6),这时参数 a=5,b=6,剩余的参数 5 和 6 组成元组传递给 c。因为参数 c 存在,所以会去执行 if 后面的语句,提示接收多余的参数 c,即(5,6),同时计算 5**6=15625。同样,power(3,3,'Hello,world')中参数 a 和 b 按顺序赋值为 3 和 3,第三个参数“Hello,world”传递给 c,也会执行 if 后面的语句,此时 c=('Hello,world',)。

**【例 5-5】** 按默认值传递参数,函数定义如下:

```
def f (x, y=None, z=1):
    if y is None:
        x, y=0, x
    result=[]
    i=x
    while i<y:
        result.append(i)
        i+=z
    return result
```

调用函数,结果如下:

```
>>>f(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>f(1,5)
[1, 2, 3, 4]
>>>f(1,10,2)
[1,3, 5,7, 9]
>>>power(*f(3,7))
接收多余的参数: (5, 6)
81
```

其中函数 power 的参数是 *f(3,7),因为 f(3,7)返回一个列表[3,4,5,6],函数调用等价于 power(3,4,5,6),根据 power 函数定义,a=3,b=4、5 和 6 组成元组传递给 *c,最后返回参数提示和 3**4 的结果 81。

# 5.3 函数的返回值

## 5.3.1 返回布尔值和列表的函数

在函数定义中，return 语句是可选项，主要功能是返回函数运行结果。返回的数据类型除常见的数值、字符串外，还可以是布尔值或列表等。

**【例 5-6】** 下面程序判断用户输入的英文单词首字母是否是大写字母，并给出提示。

```
def isBigLetter(word):
    if ord('A')<=ord(word[0])<=ord('Z'):
        return True
    return False
    str=input('请输入一个英文单词：')
    if isBigLetter(str):
        print('单词首字母是大写字母！')
    else:
        print('单词首字母不是大写字母！')
```

其中函数 isBigLetter 用来判断单词首字母是否为大写字母，return 语句返回一个布尔值，如果是大写字母返回 True，否则返回 False。函数中 word[0]表示字符串 word 中的第一个字符，也就是单词首字母。ord(x)是 Python 内置函数，用于将字符串 x 转换为其所对应的 Unicode 编码。因为大写字母 A～Z 在计算机中用编码 65～90 表示，如果字符编码在这个范围，就表示该字符是大写字母。

请注意，函数中出现了两个 return 语句。如果 if 语句条件成立，会去执行“return True”语句，然后函数结束运行，返回调用程序，不再执行“return False”语句。如果 if 语句条件不成立，会执行“return False”语句，再返回调用程序。当函数中有多条 return 语句时，执行完第一条 return 语句就会退出函数，不再执行其他 return 语句。

程序运行结果如下：

```
=======RESTART: C:/Python36-32/例 5.6.py=======
请输入一个英文单词：Python
单词首字母是大写字母！
>>>
=======RESTART: C:/Python36-32/例 5.6.py=======
请输入一个英文单词：python
单词首字母不是大写字母！
```

return 还可以返回列表。例如，如果要求显示用户输入的英文单词中包含的元音字母，可以用下面的程序：

**【例 5-7】** 显示英文单词中的元音字母。

```
def find_Vowels(word):
    word=word.lower()
    vow=('a','e','i','o','u')
    findVowels=[]
    for i in vow:
        if (i in word) and (i not in findVowels):
            findVowels.append(i)
    return findVowels
    str=input("请输入一个单词: ")
    vowelsinStr=find_Vowels(str)
    print(vowelsinStr)
```

为方便处理,函数 find_Vowels 首先使用 str.lower()方法将单词的所有字母转换为小写,再依次判断 5 个元音字母(a,e,i,o,u)是否包含在单词中。如果包含在单词中,就将其添加到列表 findVowels 中,最后返回这个列表。调用函数时需要定义一个变量保存列表,程序运行结果如下:

```
=======RESTART: C:/Python36-32/例 5-7.py=======
请输入一个单词: Gudio
['i', 'o', 'u']
```

## 5.3.2 无返回值的函数

函数中如果没有 return 语句,会自动返回 None,表示没有返回值。它的数据类型是 NoneType。例如:

```
def f(x,y):
    s=x+y
```

调用这个函数:

```
>>>result=f(2, 3)
>>>print(result)
None
>>>type(result)
<class 'NoneType'>
```

如果函数有 return 语句,但没有任何参数,则运行结果与上例相同。例如:

```
def f(x,y):
    s=x+y
    return
```

调用函数,结果如下:

```
>>>result=f(2, 3)
>>>print(result)
None
>>>type(result)
<class 'NoneType'>
```

### 5.3.3 返回多值的函数

return 语句还可以返回多个值,此时返回的数据类型实际上是一个元组。

**【例 5-8】** 下面的程序计算一个三角形的面积、周长和底边 z 的高。

```
def triangle(x, y, z):
    c=x+y+z
    p=c/2
    s=p* (p-x) * (p-y) * (p-z)
    s=pow(s, 0.5)
    hz=2* s/z
    return c, s, hz
circle, square, heightofz=triangle(10,8,6)
print('三角形的周长是:', circle)
print('三角形的面积是:', square)
print('三角形底边 z的高是:', hz)
```

函数 triangle 返回了三角形的周长、面积以及 z 边的高。调用函数时需要定义三个变量保存返回的结果。

```
=======RESTART: C:/Python36-32/例 5-8.py=======
三角形的周长是: 24.0
三角形的面积是: 24.0
三角形底边 z的高是: 8.0
```

函数返回的结果实际上是一个元组,调用函数如下:

```
>>>result=triangle(10,8,6)
>>>print(result)
(24.0, 24.0, 8.0)
```

可以发现 result 是一个元组，元组中的元素按照位置顺序同时赋值给多个变量。

## 5.4 变量的作用域

通过例 5-5～例 5-8 可以发现，如果函数中的 return 语句返回运行结果，在调用函数时必须定义变量来保存结果。这是因为 Python 内部有内存管理(memory management)机制，在创建一个函数时会为其分配内存空间，当函数运行完毕后会释放这个内存空间，相应地，函数内部创建的各种变量，包括保存运行结果的变量会随之消失。因此必须在调用函数时定义变量来保存结果。由于函数内部创建的变量只能在函数内部使用和访问，因此称为局部变量(local variable)，或者局部作用域。与之对应，如果一个变量在函数之外定义，可以被程序的任何部分(例如其他函数)访问和使用，这样的变量称为全局变量(global variable)。

**【例 5-9】** 区别函数全局变量与局部变量。

```
name='张三'
def f1():
    age=18
    name='李四'
    print (age, name)
def f2():
    age=39
    name='王五'
    print (age, name)
f1()
f2()
print(name)
print(age)
```

程序第一行定义了一个变量 name，在函数 f1()和 f2()中也分别定义了名称相同的变量 name，但这三个 name 是不同的变量。第一行中的 name 是全局变量，可以被所有函数使用；f1()和 f2()中的 name 是局部变量，仅能在各自的函数内部使用。当函数内部定义一个与全局变量同名的变量时，Python 会创建一个新的局部变量，以防止函数无意间改变全局变量。因此函数 f1()和 f2()内部的 name 值不是"张三"，而是各自定义的变量值"李四"和"王五"。函数外部的 print 语句调用的是全局变量 name，因此值为"张三"。这个值也不会被函数 f1()和 f2()修改。同样，age 也是局部变量，无法在函数外调用，因此最后一条 print 语句提示"没有定义 age"的错误信息。

程序执行结果如下：

```
=======RESTART: C:/Python36-32/例 5-9.py=======
18 李四
```

```
39 王五
张三
Traceback (most recent call last):
    File "C:/Python36-32/例 5.9.py", line 13, in<module>
      print(age)
NameError: name 'age' is not defined
```

例 5-9 展示了一种定义全局变量的方法，即在程序顶部创建全局变量的赋值语句。通过本例可以看出，全局变量不能在函数内部进行修改，局部变量不能在函数外访问，如果想在函数内部修改变量，要用关键字 global 在函数内部创建一个全局变量，形式如下：

```
global 全局变量名 函数名 (**参数)
```

修改例 5-9 的程序如下：

```
name='张三'
def f1():
    age=18
    name='李四'
    print (age, name)
def f2():
    age=39
    global name
    name='王五'
    print (age, name)
f1()
f2()
print(name)
print(age)
```

在函数 f2 中用关键字 global 定义了一个全局变量 name，并赋予新的值“王五”。此后全局变量 name 的值就会从“张三”变成“王五”，因此函数外部的 print(name)语句的执行结果是“王五”。但 age 和函数 f1()中的 name 仍然是局部变量，仅在函数内部有效，程序执行结果如下：

```
=======RESTART: C:/Python36-32/例 5-9.py=======
18 李四
39 王五
王五
Traceback (most recent call last):
    File "C:/Python36-32/例 5-9.py", line 14, in<module>
        print(age)
NameError: name 'age' is not defined
```

可以看出，global 语句仅影响所在函数内其后的语句。一般情况下，应避免使用全局变量，特别是在大型程序中，全局变量会使程序可读性下降，并且很容易引起错误。

再看下面的例子：

```
name="张三"
def f1():
    name="李四"
def f2():
    name="王五"
    print(name)
f2()
f1()
```

函数在读取变量时，优先读取函数本身自有的局部变量，再去读全局变量。因此函数 f2 的 prin(name)语句中的 name 值应为"王五"，程序运行结果是"王五"。

如果程序如下：

```
name="张三"
def f1():
    print(name)
def f2():
    name="李四"
f1()
f2()
```

运行结果：

```
张三
```

函数 f2 创建了局部变量 name"李四"，并调用函数 f1。函数 f1 的功能是显示变量 name，由于函数自身没有定义这个变量，f1 会到函数外部调用全局变量 name"张三"，因此程序的运行结果是"张三"。

# 5.5 递　　归

## 5.5.1 递归的定义

函数可以调用其他函数，也可以调用自身，这样的函数称为递归函数。几乎在所有编程语言中，递归都是一种重要的编程方法。递归常用来解决结构相似的问题，其基本思路是将一个复杂问题转化成一个或几个子问题，子问题的形式和结构与原问题相似，但规模

更小，再把这些子问题进一步分解成规模更小的子问题，直到每个子问题可以直接求解。因此，递归有两个基本要素：

（1）基例：确定递归何时终止，即何时终止分解子问题，也称为递归出口。

（2）递归模式：即复杂问题如何分解为子问题，也称为递归体。

递归函数只有具备了这两个要素，才能在有限次计算后得出结果，例如，经典的递归实例——阶乘。已知整数 n 的阶乘计算公式是"n！＝1×2×3×…×n"。我们观察阶乘公式可以发现，计算 n！可以转化为计算 n＊（n－1）！这个子问题，如果用函数 fact(n)表示 n！，fact(n)＝n＊fact(n－1)，fact(n－1)与 fact(n)相比，形式相同（都是求阶乘），规模变小（求 n－1 的阶乘）；fact(n－1)可以分解为规模更小的 fact(n－2)＊（n－1），fact(n－2)可以继续分解为 fact(n－3)＊（n－2），……。当分解到 fact(0)时，0！＝1，此时停止递归，这就是递归出口，也就是分解子问题直到问题可以直接求解为止。递归体就是 n＊fact(n－1)，因此阶乘还有另外一个递归定义：

（1）0 的阶乘是 1（递归出口）。

（2）n！＝n＊（n－1）！（递归体）。

由于阶乘运算具备递归的两个基本要素，可以用递归形式定义函数。

**【例 5-10】** 计算并输出整数 n 的递归值。

```
def fact(n):
    if n==0:
        return 1
    else:
        return n* fact(n-1)
num=int(eval(input("请输入一个整数:")))
print(fact(abs(num)))
```

运行结果：

```
请输入一个整数:10
3628800
```

当然这个函数也可以用循环来实现：

```
def fact(n):
    result=n
    for i in range(1, n)
        result*=i
    return result
num=int(eval(input("请输入一个整数:")))
print(fact(abs(num)))
```

理论上，所有的递归函数都可以写成循环的方式，但是递归定义简单，逻辑清晰。如

果理解递归函数的定义，递归通常更加易懂，能明显提高程序的可读性。

通过例 5-10 可以总结递归的一般定义形式如下：

```
def  递归函数：
    if  基例：
        return  基例结果
    else:
        return  递归体
```

如果一个函数不具备递归的两个基本元素，由于每次调用函数都会占用计算机的一部分内存，当函数调用多次后，程序会返回“超过最大递归深度”的错误信息。这类递归称为无穷递归，类似于 while true 的无穷循环，中间没有 break 和 continue 语句。

## 5.5.2 递归实例

下面分析两个经典的递归实例。

### 1. 幂运算

整数幂运算 $y=x^n$ 也符合递归的两个条件：当 $n=0$ 时，$y=x^0=1$，这是递归出口；其他情况下 $y=x^n=x*x^{n-1}$，这是递归体。递归形式函数定义如下：

**【例 5-11】** 计算并输出 x 的 n 次幂（n 为整数）。

```
def power(x, a):
    if a==0:
        return 1
    else:
        return x* power(x, a-1)
base, index=eval(input("请输入底数和指数:"))
print(power(base,index))
```

运行结果：

```
请输入底数和指数:10,3
1000
请输入底数和指数:10,0
1
请输入底数和指数:0,0
1
```

### 2. 二分法查找

被提问者默想一个 1～100 的数字，提问者通过提问来猜这个数字。被提问者每次回

答问题只能用“是”或“不是”,提问者需要提问多少次可以猜中? 答案是 7 次。这个问题的实质是查找一个排序序列中的数字。以序列中点为界将序列分为两个部分(二分法的名字由此而来),通过询问数字在序列的哪个部分来逐步缩小查找范围,最终锁定数字的位置。例如 1～100 的数字序列,以中点 50 为界分为两部分。第一次问“数字是否大于 50?”,如果被提问者回答说“是”,则小于 50 的一半数字被淘汰,查找范围缩小至 50～100,这个数字序列的中点是 75(序列上下边界的平均值),因此接着问“数字是否大于 75?”,如果回答“是”,则将查找范围进一步缩小至 75～100,然后继续提问直至找出满足条件的数字为止。很明显二分法查找也是一种递归形式:每次递归会缩小一半查找范围,当查找范围的上下边界相同时,这个边界值就是要猜的数字,也就是递归出口;否则以查找范围的中点(上下边界的平均值)为界,继续缩小查找范围,查找数字所在的位置。这个递归例子的关键是顺序。

**【例 5-12】** 猜数字游戏——二分法查找的递归程序。

```
def search(lower, upper, num):
    if lower==upper:
        print("我猜这个数字是", upper, "!")
    else:
        midpoint=(lower+upper)//2
    if num>midpoint:
        return search(midpoint+1, upper, num)
    else:
        return search(lower, midpoint, num)
seqlower, sequpper=eval(input("要猜的数字所在范围:"))
number=eval(input("默想的数字是:"))
search(seqlower, sequpper, number)
```

运行结果:

```
要猜的数字所在范围:1,100
默想的数字是:34
我猜这个数字是 34 !
要猜的数字所在范围:1,1000
默想的数字是:783
我猜这个数字是 783 !
```

# 本 章 小 结

(1) 了解函数的优点和特性。
(2) 掌握函数的基本定义和调用方法。
(3) 掌握常用的函数参数传递方法。

(4) 了解可变数量的参数传递。
(5) 理解函数的作用域。
(6) 理解递归函数的设计方法。

# 习　　题

## 1. 简答题

(1) 简述使用函数的原因和意义。
(2) 什么是匿名函数？请举例说明。
(3) 什么是递归函数？请举例说明。
(4) 使用哪个关键字创建函数？
(5) 如何调用函数？
(6) 如何向函数传递参数？
(7) 函数最多可以有多少个参数？
(8) 如何从函数返回结果？
(9) 下面的函数有返回值吗？

```
def Hello():
    print("Hello Python")
```

(10) 函数运行结束后，函数中的局部变量会发生什么变化？
(11) 如果希望在函数中修改全局变量的值，需要使用什么关键字？
(12) 递归必须满足什么条件？
(13) 下列程序的输出结果是什么？

```
x=7
def main():
    x=5
    f()
    print(x)
def f():
    print(x)
main()
```

```
x=7
def main():
    global x
    x=5
    f()
    print(x)
main()
```

## 2. 填空题

(1) 在函数内部可以通过关键字________来定义全局变量。
(2) 一个没有 return 语句的函数的返回结果是________。
(3) return [1,2],{3,4},5 语句返回的类型是________。

(4) 请使用 lambda 表达式表示下列函数________。

```
def fun(x, y=4)
    return y* x
```

### 3. 选择题

(1) 函数的实际参数可以是【　　】。

A) 变量　　B) 常量　　C) 函数　　D) 以上都行

(2) 函数 isinstance(1+0j,int)返回的结果是【　　】。

A) True　　B) False　　C) SyntaxError　　D) TypeError

(3) 下列函数中,可以用于将数字转换成字符的是【　　】。

A) ord()　　B)chr()　　C) oct()　　D) hex()

(4) 下列语句中,正确的是【　　】。

A) def f(a=0,b):　　B) def f(a,b==0):

C) def f(a,b, * ):　　D) def f(a, * b):

(5) 执行下列程序段返回的结果是【　　】。

```
def f(* a):
    print(type(a))
    if f(9, 9):
        True
    else:
        None
```

A) True　　B) False

C) None　　D) <class 'tuptle'>

(6) 下列定义的匿名函数能够返回两个数中较大的是【　　】。

A) fmax=lambda x,y:x if x>y else y

B) fmax=lambda x,y:x if: x>y else: y

C) fmax=lambda x,y: if x>y,x else y

D) fmax=lambda x,y: if: x>y,x else: y

(7) 下列程序段返回的结果是【　　】。

```
a="first"
def second(a):
    a="second"
def third():
    global a
a="third"
```

```
third()
print(a,end=',')
second("forth")
print(a)
```

A) first,second　　B) second,third　　C) third,forth　　D) second,first

(8) 下列关于函数的说法正确的是【　　】。

A) 函数定义时必须有形参　　B) 函数定义时必须有 return 语句

C) 函数的形参和实参应该同名　　D) 以上都是错的

(9) 若匿名函数 f=lambda x,y: x+y,则 f({1,2},{3,4})返回的结果是【　　】。

A) {1,2,3,4}　　B) {4,6}　　C) TypeError　　D) SyntaxError

(10) 若匿名函数 f=[lambda x=3: x * 3,lambda x: x * * 3],则 f[1](f[0]())返回的结果是【　　】。

A) 728　　B) 729　　C) TypeError　　D) SyntaxError

(11) 若 def f(): pass,则 type(f())返回的结果是【　　】。

A) <class 'NoneType'>　　B) <class 'bool'>

C) <class 'function'>　　D) 以上都不是

(12) 下列语句中返回值为 5.0 的是【　　】。

A) math.ceil(4.2)　　B) math.floor(5.3)

C) round(5.3)　　D) 以上都不是

(13) 若匿名函数 f=[lambda x=2: x * 2,lambda x=2: x**2],则 f[2](f[1]())返回的结果是【　　】。

A) 16　　B) 24　　C) IndexError　　D) SyntaxError

(14) 执行下列程序段返回的结果是【　　】。

```
def f(* a): print(a)
q= [1,2,3,5,9]
f(* q)
```

A) (1,2,3,5,9)　　B) [1,2,3,5,9]　　C) SyntaxError　　D) TypeError

### 4. 程序设计题

(1) 定义 Max 函数返回一个数字列表中的最大值。

(2) 定义函数,测试输入的字符串是否为回文联(回文联,即用回文形式写的对联,顺读倒读内容完全一样,如“山果花开花果山”)。

(3) 使用递归实现上题中的函数。

(4) 使用递归编写一个将十进制转换为二进制的函数(采用“除 2 取余”的方式,结果返回字符串形式)。

(5) 有两种薪酬计算方案,请确定哪种薪酬方案更好。方案 1: 每天 100 元。方案 2:

第一天 100 元，第二天 200 元，第三天 400 元，每天的金额增长一倍，以此类推。使用名为 option1 和 option2 的函数计算在两种方案下的收入（假设工作 10 天）并输出结果。

（6）编写程序，要求输入一个人的姓名和目前的年薪，然后计算这个人下一年的薪水。如果年薪＜40000 元，则下一年的年薪将增长 5％。如果年薪≥40000 元收入，下一年的年薪除增加 2000 元外，还会增加超过 40000 元部分的 2％。使用函数处理输入、输出和计算新的薪水并输出，形式如下：

```
请输入姓名：**
请输入目前的年薪：****元
* * 新一年的年薪是：****元
```

（7）美国公务员退休制度中，一个人服务至少 20 年后可以在 55 岁退休。一个简单的计算退休金的方法如下：

① 计算收入最高的三年的年平均薪水，记为 ave。

② 计算月数/12，记为 yrs。

③ 计算比例：头 5 年 1.5％，接下来 5 年 1.75％，之后每年 2％，记为 perRate。

④ 取 perRate 和 80％中的较小值，记为 p。

⑤ 退休金金额为：p * ave。

编写一个程序，要求输入如下，并计算退休金金额。ave 和 p 的值应该通过函数来计算。

```
请输入您的年龄：**
请输入您的工作年限：**
请输入三个最高年薪中的第一个：****
请输入三个最高年薪中的第二个：****
请输入三个最高年薪中的第三个：****
您的退休年年薪是：****
```

（8）英文中表示月份的 12 个单词中，如果包含字母 r，则称这个月份为 R 月。现有一个 Months.txt 文件，包含 12 行，每行有一个表示月份的单词。编写程序显示包含有字母 r 的月份。要求程序使用全局变量 months，并初始化为空列表。main 函数应该调用三个函数，一个用于使用文本文件中的内容填充 months 列表，一个用于从 months 列表中删除不包含字母 r 的月份，一个用于显示仍然留在列表中的月份名。

（9）威尔逊定理。如果一个数的因子只有 1 和它本身，这个数字即为素数。编写一个程序，通过使用定理“一个数字 n 是素数当且仅当(n－1)!＋1 能被 n 整除”来判断一个数字是否是素数。程序应该定义一个返回布尔值的名为 isPrime 的函数，并调用名为 factorial 的函数。

# 第 6 章　面向对象的编程方法

到目前为止，在我们的程序中，都是根据所处理问题的过程为中心来编写程序的，将需要使用的函数或语句进行有序堆积，这称为面向过程的编程。还有一种把数据和功能结合起来，用称为“对象”的架构包裹起来组织程序的方法，这称为面向对象的编程。在大多数时候我们可以使用面向过程的编程方法，但当编写大型软件或寻求一个更加有效的、更适合描述问题空间的解决方案时候，应该使用面向对象的编程技术。

## 6.1　面向对象基础知识

### 6.1.1　对象与面向对象

高级程序设计语言中最基本的构成要素是变量（数据）、表达式、语句和函数，通过算法对这些元素进行组合可以实现具有复杂功能的程序。前面章节的程序设计方法为面向过程的程序设计。面向过程就是分析出解决问题所需要的步骤，将这些步骤编写成函数模块，将一个函数调用就可以实现一个步骤，使用的时候依次调用这些函数，就可完成程序任务。比如说玩游戏时候，需要经历登录界面、输入密码、选择角色、游戏玩要和结束游戏几个步骤。用面向过程的程序设计方法设计游戏时，程序包括游戏登录模块、密码输入模块、角色设置模块、游戏玩要模块和游戏结束模块。程序依次调用这些模块就能完成程序功能。面向过程的程序设计方法没有将客体看作一个整体，比如游戏设计中，没有把所有的玩家看作一个整体，而是以功能为目标来设计构造应用程序。用这种思想进行程序设计，没有遵循人类观察问题和解决问题的基本思路，而且增加了程序设计的复杂程度。比如说学生起床上学要经历起床、穿衣服、洗脸刷牙、去上学四个步骤，面向过程程序设计没有把学生看成一个整体。因为起床、穿衣、洗脸刷牙、去上学是学生这个类早晨要做的四件事情，是和学生这个类相关的四件事情，在程序编写时，可以把客体学生及其学生所具有的属性、所能做的事情看成一个整体进行编写，这样就会简化程序，这种把客体看成整体的编程思路就是面向对象的编程方法。

面对程序规模的日益扩大、运行环境的日趋复杂、需求的不断变化，将算法和程序设计的基本方法统一到人类解决问题的思维方法之上的需求日益强大，面向对象的程序设计方法（Object-oriented Programming，OOP）因此被提了出来。面向对象方法的出发点和基本原则是尽可能地模拟现实世界中人类的思维方式，使问题求解方法和过程尽可能地接近人类解决现实问题的方法和过程，把构成问题的事务分解成各个对象，建立对象的

目的不是为了完成一个步骤，而是为了描述某个事物在整个解决问题的步骤中的行为。面向对象程序设计是面向问题中的各种独立个体的，程序的分析设计过程就是将程序分解成不同对象之间的交互的过程。这就好比在针对某个工程或游戏设计程序时先不考虑游戏是怎么玩的，工作是怎么做的，而先会去确定游戏或工程中有哪些人或事物参与（一般选择、用户、玩家、角色等），然后再看他们都有什么用，都干了些什么，针对这个区别设计方法。最后在通过这些千丝万缕的联系把它们分门别类地组装在一起。例如电子宠物游戏程序，每个电子宠物都是一个对象，每个游戏角色都是一个对象，每个对象都有各自特性，电子宠物需要被喂养，玩家有喂养宠物功能，电子宠物一段时间没有进食会生病，玩家需要花钱给电子宠物买食物，带电子宠物在游戏中玩，让电子宠物健康成长。游戏的设计都是面向角色对象的。这种游戏的虚拟世界和现实非常接近，程序员就是用面向对象的编程方法进行游戏设计。

Python 中一切皆为对象：我自己就是一个对象，我玩的计算机就是对象，坐着的椅子就是对象，家里养的小狗也是一个对象……我们通过描述属性（特征）和行为来描述一个对象的。比如家里的小狗，它的颜色、大小、年龄、体重等是它的属性或特征。它会汪汪叫、会摇尾巴等是它的行为。我们在描述一个真实对象（物体）时包括两个方面：它可以做什么（行为），它是什么样的（属性或特征）。在 Python 中，一个对象的特征称为属性（attribute），它所具有的行为称为方法（method）。所以说，对象是将数据（属性）和操作（行为方法）两者结合在一起而抽象出的一种实体。对象拥有一些数据，同时也知道如何对这些数据进行操作。

例如，张三同学可视为一个对象：

（1）张三具有自己的数据，如姓名、出生日期、身高等。

（2）张三对他的数据都有自己的操作方法，例如，通过计算当前日期与出生日期的差值来得到年龄等。

（3）张三都能响应外部发来的消息（如询问年龄的消息），也就是执行相应的数据操作。

（4）“张三”和“李四”是两个不同对象，但如果他们具有共同的属性，如学号、姓名、性别、身高、体重等，可以把他们的共同特性抽象出来，说他们是学生。在程序设计时，可以抽象出来一个“学生”的类。“张三”和“李四”分别是“学生”类的实例，是两个独立的对象。

### 6.1.2 类

把众多的事物按其一定的属性归纳划分为不同的类别，是人类在认识客观世界时经常采用的思维方法。把具有共同性质的事物划分为一类，得出一个抽象的概念。例如，马、树木、石头等都是一些抽象概念，它们是一些具有共同特征的事物的集合，称为类。类的概念使我们能对属于该类的全部个体事物进行统一的描述。例如，树具有树根、树干、树枝和树叶，它能进行光合作用，这个描述适合所有的树，从而不必对每棵具体的树进行一次这样的描述。世界上所有的汽车归为汽车类，所有的动物归为动物类。汽车类有共同的状态（引擎数、挡位数、颜色、轮胎型号等）和行为（换挡、开灯、刹车等）。

在面向对象的方法中，类的定义是：类是具有相同属性和服务功能的一组对象的集合，它为该类的全部对象提供了统一的抽象描述，其内部包括属性和服务两个主要部分。在面向对象的编程语言中，类是一个独立的程序单位，它应该有一个类名并包括属性(数据)定义和行为定义两个主要部分。

类与对象的关系，如同一个模具与用这个模具铸造出来的铸件之间的关系。类给出了属于该类的全部对象的抽象定义，而对象则是符合这种定义的一个实体。所以，一个对象又称为类的一个实例(Instance)。

类的变量由一个类的所有对象(实例)共享使用，即所有对象共享一个变量地址，所以当某个对象对类的变量做了改动时，这个改动会反映到所有其他的实例上。

对象的变量由类的每个对象(实例)独立拥有，因此每个对象有自己对这个变量的一份副本，即它们不是共享的。在同一个类的不同实例中，虽然对象的变量有相同的名称，但是互不相关的。

类具有如下一些特性。

### 1. 封装性

封装是面向对象方法的一个重要原则，它有两个含义。第一个含义是，把对象的全部属性和全部行为结合在一起，形成一个不可分割的独立单位(即对象)；第二个含义也称为"信息隐蔽"，即尽可能隐蔽对象的内部细节，对外形成一个边界(或者说形成一道屏障)，只保留有限的对外接口，使之与外部发生联系。例如，人要在黑板上画圆，这一共涉及三个对象：人、黑板、圆，画圆的方法要分配给哪个对象呢？由于画圆需要使用到圆心和半径，圆心和半径显然是圆的属性，如果将它们在类中定义成了私有的成员变量，那么，画圆的方法必须分配给圆，它才能访问到圆心和半径这两个属性，人以后只是调用圆的画圆方法、表示给圆发个消息而已，画圆这个方法不应该分配在人这个对象上，这就是面向对象的封装性，即将对象封装成一个高度自治和相对封闭的个体，对象状态(属性)由这个对象自己的行为(方法)来读取和改变。再举个例子，司机将火车刹住了，刹车的动作是分配给司机，还是分配给火车，显然，应该分配给火车，因为司机自身是不可能有那么大的力气将一个火车给停下来的，只有火车自己才能完成这一动作，火车需要调用内部的离合器和刹车片等多个器件协作才能完成刹车这个动作，司机刹车的过程只是给火车发了一个消息，通知火车要执行刹车动作而已。这两个例子中圆和画圆方法，火车和刹车动作可以封装在一起。

### 2. 继承性

面向对象的编程带来的主要好处之一是代码的重用。实现这种重用的方法之一是通过继承机制。假设，我们想要写一个程序来记录学校教师和行政人员情况。他们有一些共同属性，比如姓名、年龄和地址。他们也有专有的属性，比如教师的课程等。你可以为教师和行政人员建立两个独立的类来处理，但是这样做的话，如果要增加一个新的共有属性，就意味着要在这两个独立的类中都增加很多重复的属性，很快就会发现这样很不实

用。一个比较好的方法是创建一个共同的类，称为“员工”，然后让“教师”和“行政人员”的类继承这个共同的类。即它们都是这个类型（类）的子类型，然后再为这些子类型添加专有的属性。

使用这种方法有很多优点。如果增加或改变了“员工”类中的任何功能，它会自动地反映到子类型之中。例如，你要为“教师”和“行政人员”都增加一个新的身份证属性，那么只需简单地把它加到“员工”类中。然而，在一个子类型之中做的改动不会影响到别的子类型。在上述的场合中，“员工”类称为父类，而“教师”和“行政人员”类称为子类。

### 3. 多态性

多态性是指在父类中定义的属性或行为被子类继承之后，可以具有不同的数据类型或表现为不同的行为。这使得同一个属性或行为在父类及其各个子类中可以具有不同的语义。

多态性为程序开发者带来不少方便。例如，有一个“几何图形”的类，定义了数学中几何图形的属性和服务，“椭圆”“多边形”“正方形”等类都可以从“几何图形”类继承得到。在父类“几何图形”中定义了一个服务“绘图”，但并不确定执行时到底绘制一个什么图形。子类“椭圆”和“多边形”都继承了几何图形类的绘图服务，但其功能却不同：一个是绘制一个椭圆，一个是绘制一个多边形。进而，在多边形类更下层的子类“矩形”中绘图服务又可以采用一个比绘制一般的多边形更高效的算法来绘制一个矩形。这样，当系统的其余部分请求绘制任何一种几何图形时，消息中给出的服务名同样都是“绘图”，而椭圆、多边形、矩形等类的对象接收到这个消息时却各自执行不同的绘图算法。

## 6.1.3 面向对象的程序设计

类和对象是面向对象编程的两个主要方面：类创建一个新类型，而对象是这个类的实例。类似于 C 语言中 int 类型的变量，这种存储整数的变量是 int 类的实例（对象）。

### 1. 类的定义

在 Python 语言中，类使用 class 关键字来创建。程序通过 class 告诉系统要定义一个类，让系统用理解类的规则来理解随后的代码。类的域和方法被列在一个缩进块中。一个尽可能简单的类如下面这个例子所示。

**【例 6-1】** 创建一个类。

```
class Person:
    print('这是一个类：Person')

p= Person()
print(p)
```

输出结果：

```
这是一个类：Person
<__main__.Person object at 0x0000013D23F60CF8>
```

在 class 语句的后面写上类名，就创建了一个新类。其后跟着一个缩进的语句块形成类体。在这个例子中，使用了一个打印语句 print，它打印了一句话“这是一个类：Person”。

接下来，使用类名后跟一对圆括号来创建一个对象（实例）。为了验证，我们简单地打印了这个变量的类型。它告诉我们已经在主程序中有了一个 Person 类的实例。

可以注意到存储对象的计算机内存地址也打印了出来。这个地址在你的计算机上可能会是另外一个值，因为 Python 可以在任何空位存储对象。

## 2. 对象方法的定义

类可以拥有像函数一样的方法，这些方法与函数是作为类生成的对象所拥有的行为来使用的。现在来学习一个例子。

**【例 6-2】** 使用对象的方法。

```
class Person:
    name='小明'
    def sayHi(self):
        print('Hello, how are you? ')
        print('我的名字叫'+self.name)

p=Person()
p.sayHi()
```

输出结果：

```
Hello,how are you?
我的名字叫小明
```

这里看到了 self 的用法。self 指的是类实例（对象）本身（注意：不是类本身）。在上述例子中，self 指向 Person 的对象 p。为什么不是指向类本身呢，如例 6-3 所示，Person 的对象有两个 p1 和 p2，如果 self 指向类本身，那么当有多个实例（对象）时，self 指向哪一个呢？所以，self 指向类对象。self. name 表示 name 是类所拥有的属性变量。sayHi 是对象方法，当 sayHi 方法被调用时，对象会将自身 self 作为第一个参数传入。所以在定义时，需要将 self 写入“def sayHi(self)”。

## 3. 使用__init__方法

在 Python 语言中，类中有很多方法的名字具有特殊的重要意义。现在我们将学习

__init__方法的意义。__init__方法在类的一个对象被建立时，马上运行。这个方法可以用来对对象做一些你希望的初始化。注意，这个名称的开始和结尾都是双下画线。

**【例 6-3】** 使用__init__方法。

```
class Person:
    def __init__(self, name):
        self.name= name
    def sayHi(self):
        print('Hello, 我的名字是', self.name)

p1= Person('小红')
p2= Person('小白')
p1.sayHi()
p2.sayHi()
```

输出结果：

```
Hello, 我的名字是小红
Hello, 我的名字是小白
```

__init__方法类似于 C++ 、C＃ 和 Java 中的构造函数。这里，我们把__init__方法定义为取一个参数 name(以及普通的参数 self)。在这个__init__里，只是创建一个新的域，也称为 name。注意它们是两个不同的变量，尽管它们有相同的名字，句点使我们能够区分它们。

最重要的是在程序中没有专门调用__init__方法，只是在创建一个类的新实例的时候，把参数包括在圆括号内跟在类名后面，从而传递给__init__方法。这是这种方法的重要之处。现在，我们能够在程序方法中使用 self. name 属性变量。这在 sayHi 方法中得到了验证。

### 4. 类与对象

类是对具有相同数据和方法的一组对象的描述或定义，而对象就是一个类的实例。

**【例 6-4】** 使用类与对象的变量。

```
class Person:
    population= 0
    def __init__(self, name):
        self.name= name
        print('(初始化... %s)' %self.name)
        Person.population+ =1

    def __del__(self):
        print('%s 走了.' %self.name)
        Person.population- =1
```

```
        if(Person.population==0):
            print('我是最后一个人.')
        else:
            print('还有 %d 个人在线.' %Person.population)

    def sayHi(self):
        print('大家好,我的名字是 %s.' %self.name)

    def howMany(self):
        if(Person.population==1):
            print('只剩下我一个人了.')
        else:
            print('一共有 %d 个人在这里.' %Person.population)

ming=Person('小明')
ming.sayHi()
ming.howMany()

hong=Person('小红')
hong.sayHi()
hong.howMany()

ming.sayHi()
ming.howMany()
```

第一次运行的输出结果:

```
(初始化... 小明)
大家好,我的名字是小明.
只剩下我一个人了.
(初始化... 小红)
大家好,我的名字是小红.
一共有 2 个人在这里.
大家好,我的名字是小明.
一共有 2 个人在这里.
```

在这里,population 属于 Person 类,因此是一个类的变量。name 变量属于对象(它使用 self 赋值),因此是对象的变量。

观察可以发现,__init__方法用一个名字来初始化 Person 实例。在这个方法中让 population 增加 1,这是因为我们增加了一个人。同样可以发现,self. name 的值是根据每个对象指定的,这表明了它作为对象变量的本质。

就如同__init__方法一样,还有一个特殊的方法__del__,__del__方法与析构函数的

概念类似。它在对象销毁的时候被调用。对象销毁即对象不再被使用,它所占用的内存将返回给系统。在这个方法里面,程序只是简单地把 Person. population 减 1。

当对象不再被使用时,__del__方法运行,但是很难保证这个方法究竟在什么时候运行。如果你想要指明它的运行,就得使用 del 语句。

### 5. 使用继承

继承是面向对象程序设计中的一个重要概念,即如果一个类 A 继承了另一个类 B 的全部特征,就把 A 称为 B 的子类,而把 B 称为 A 的父类。使用继承的示例如例 6-5 所示。

**【例 6-5】** 使用继承示例。

```
class SchoolMember:
    def __init__(self, name, age):
        self.name= name
        self.age= age
        print('(初始化对象 SchoolMember: %s)' %self.name)

    def tell(self):
        print('姓名:"%s" 年龄:"%s"' %(self.name, self.age))

class Teacher(SchoolMember):
    def __init__(self, name, age, salary):
        SchoolMember.__init__(self, name, age)
        self.salary= salary
        print('(初始化 Teacher: %s)' %self.name)

    def tell(self):
        SchoolMember.tell(self)
        print('工资: "%d"' %self.salary)

class Student(SchoolMember):
    def __init__(self, name, age, marks):
        SchoolMember.__init__(self, name, age)
        self.marks= marks
        print('(初始化 Student: %s)' %self.name)
    def tell(self):
        SchoolMember.tell(self)
        print('成绩: "%d"' %self.marks)

t= Teacher('刘老师', 40, 30000)
s= Student('小明同学', 22, 75)

members= [t, s]
for member in members:
    member.tell()
```

输出结果：

```
(初始化对象 SchoolMember: 刘老师)
(初始化 Teacher: 刘老师)
(初始化对象 SchoolMember: 小明同学)
(初始化 Student: 小明同学)
姓名:"刘老师" 年龄:"40"
工资: "30000"
姓名:"小明同学" 年龄:"22"
成绩: "75"
```

为了使用继承，我们把基本类的名称作为一个参数放在定义类时的类名称之后。可以注意到，基本类的__init__方法专门使用 self 变量调用，这样就可以初始化对象的基本类部分。注意，在使用 SchoolMember 类的 tell 方法的时候，我们把 Teacher 和 Student 的实例仅仅作为 SchoolMember 的实例。

另外，在这个例子中，程序调用了子类型的 tell 方法，而不是 SchoolMember 类的 tell 方法。可以这样来理解，Python 总是首先查找对应类型的方法，在这个例子中就是如此。如果它不能在导出类中找到对应的方法，它才开始到基本类中逐个查找。基本类是在类定义的时候，在参数之中指明的。

## 6.2　利用 turtle 库绘制图形

turtle 是 Python 3.X 引入的一个简单的绘图工具，称为海龟绘图(Turtle Graphics)。它是一个简单的绘图工具，可以把它理解为一个机器人，但只能听得懂有限的指令。

使用海龟绘图，首先需要导入 turtle，Python 程序员构建了一个库(library，库就是可以重用的代码的一个集合)，来帮助其他程序员在 Python 中使用海龟作图。当我们输入了 import turtle，就表示我们的程序能够使用 turtle 中的代码。这在程序设计中是非常方便的事情。使用如下代码导入 turtle：

```
import turtle
```

海龟画图有位置属性、方向属性和画笔属性。画布就是 turtle 为我们展开用于绘图的区域，可以设置它的大小和初始位置。在画布的坐标轴原点上，默认有一只面朝 x 轴正方向小海龟。这里我们描述小海龟时使用了两个词语：坐标原点(位置)，面朝 x 轴正方向(方向)，在 turtle 绘图中，就是使用位置方向描述小海龟(画笔)的状态。例如，小海龟是向哪个方向爬，就是向哪个方向画线。此外，海龟有画笔属性，也就是画笔的颜色、画线的宽度等，这些都是可以设定的。

绘图窗口的原点(0,0)在正中间。默认情况下，海龟向正右方移动。操纵海龟绘图有

着许多的命令，这些命令可以划分为三种，分别为运动命令(如表 6-1 所示)、画笔控制命令(如表 6-2 所示)和全局控制命令(如表 6-3 所示)。

**表 6-1　运动命令**

| 函　　数 | 说　　明 |
|---|---|
| turtle.forward(distance) | 向前移动距离 distance 代表距离 |
| turtle.backward(distance) | 向后移动距离 distance 代表距离 |
| turtle.right(degree) | 相对当前画笔方向向右转动 degree 度 |
| turtle.left(degree) | 相对当前画笔方向向左转动 degree 度 |
| turtle.setheading(degree) | 相对 x 轴正方向，海龟逆时针朝向转动 degree 度 |
| turtle.goto(x,y) | 将画笔移动到坐标为(x,y)的位置 |
| turtle.speed(speed) | 画笔绘制的速度范围[0,10]整数 |
| turtle.pendown() | 放下画笔，移动时绘制图形，缺省时也为绘制 |
| turtle.penup() | 提起笔移动，不绘制图形，用于另起一个地方绘制 |
| turtle.circle(radius, extent) | 绘制一个圆形，其中 radius 为半径，extent 为度数，例如若 extent 为 180，则画一个半圆；如要画一个圆形，可不必写第二个参数 |
| turtle.setx(a) | 将当前 x 轴移动到指定位置(垂直方向移动) |
| turtle.sety(b) | 将当前 y 轴移动到指定位置(水平方向移动) |
| turtle.home() | 设置当前画笔位置为原点，朝向东，即 x 轴正向 |
| turtle.dot(radius,color) | 绘制一个指定直径和颜色的圆点 |

**表 6-2　画笔控制命令**

| 函　　数 | 说　　明 |
|---|---|
| turtle.fillcolor(colorstring) | 绘制图形的填充颜色 |
| turtle.color(color1, color2) | 同时设置 pencolor=color1, fillcolor=color2 |
| turtle.filling() | 返回当前是否在填充状态 |
| turtle.begin_fill() | 准备开始填充图形 |
| turtle.end_fill() | 填充完成 |
| turtle.hideturtle() | 隐藏画笔的 turtle 形状 |
| turtle.showturtle() | 显示画笔的 turtle 形状 |
| turtle.fillcolor(colorstring) | 绘制图形的填充颜色 |

**表 6-3　全局控制命令**

| 函　　数 | 说　　明 |
|---|---|
| turtle.clear() | 清空 turtle 画的笔迹 |
| turtle.reset() | 清空窗口，重置 turtle 状态为起始状态 |
| turtle.undo() | 撤销上一个 turtle 动作 |
| turtle.isvisible() | 返回当前 turtle 是否可见 |
| turtle.stamp() | 复制当前图形 |
| turtle.write(s[,font=("font-name",font_size,"font_type")]) | 写文本，s 为文本内容，font 是字体的参数，里面分别为字体名称、大小和类型；font 为可选项，font 的参数也是可选项 |

下面通过几个例子来学习绘图功能，以下代码可以自己对照表中的说明进行解读。

**【例 6-6】**　画一个边长为 60 的三角形。

```
import turtle
a=60
turtle.forward(a)
turtle.left(120)
turtle.forward(a)
turtle.left(120)
turtle.forward(a)
turtle.left(120)
```

程序运行的结果如图 6-1 所示。

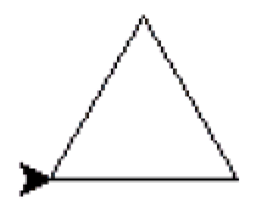


图 6-1　例 6-6 程序运行结果

**【例 6-7】**　画一个边长为 60 的正方形，并填充为红色，边框为蓝色。

```
import turtle
turtle.reset()
a=60
turtle.fillcolor("red")
turtle.pencolor("blue")
turtle.pensize(10)
turtle.fill(True)
turtle.left(90)
```

```
turtle.forward(a)
turtle.left(90)
turtle.forward(a)
turtle.left(90)
turtle.forward(a)
turtle.left(90)
turtle.forward(a)
turtle.fill(False)
```

**【例 6-8】** 绘制正方形。

```
import turtle
import time
#定义绘制时画笔的颜色
turtle.color("purple")
#定义绘制时画笔的线条的宽度
turtle.Pen.width=5
#定义绘图的速度
turtle.speed(10)
#以 0,0 为起点进行绘制
turtle.goto(0,0)
#绘出正方形的四条边
for i in range(4):
    turtle.forward(100)
turtle.right(90)
#画笔移动到点(-150,-120)时不绘图
turtle.up()
turtle.goto(-150,-120)
#再次定义画笔颜色
turtle.color("red")
#在(-150,-120)点上打印"Done"
turtle.write("Done")
time.sleep(3)
```

运行结果如图 6-2 所示。

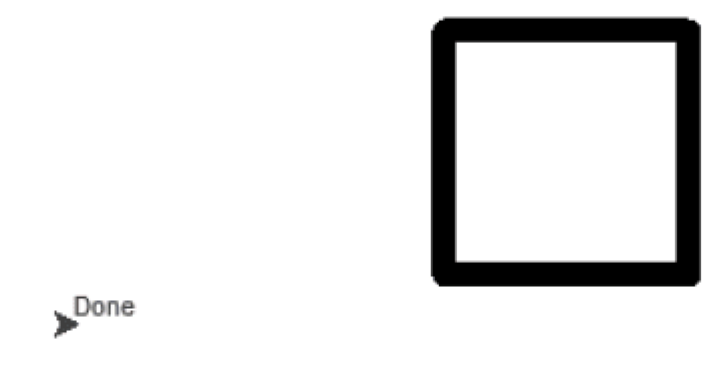



图 6-2　例 6-8 程序结果

**【例 6-9】** 绘制五角星。

```
import turtle
import time
turtle.color("purple")
turtle.Pen.width=5
turtle.goto(0,0)
turtle.speed(10)
for i in range(6):
    turtle.forward(100)
turtle.right(144)
turtle.up()
turtle.forward(100)
turtle.goto(-150,-120)
turtle.color("red")
turtle.write("Done")
time.sleep(3)
```

程序的运行结果如图 6-3 所示。

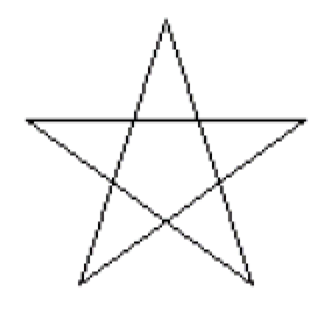



图 6-3　例 6-9 程序结果

# 6.3　Python 科学计算

现在,Python 语言广泛应用于科学计算,学习 Python 不能不学习如何使用它进行科学计算。下面介绍几个常用的 Python 科学计算工具包,其中,NumPy 是一个定义了数值数组和矩阵类型及其基本运算的语言扩展,Matplotlib 是一个绘制科学图表的语言扩展。SciPy 是另一种用 NumPy 来实现高等数学、信号处理、优化、统计和许多其他科学任务的语言扩展。

python 程序员编写了用于各种科学计算的程序,并且把这些程序分门别类放到库里,我们只需要安装这些第三方数据库,就可以直接使用 Python 程序员编写的代码命令来处理我们的数据,不必自己编写程序。

## 6.3.1 NumPy 处理数据

NumPy 是一种高性能科学计算和数据分析的基础包。利用 NumPy 包可以扩充 Python 的数据处理能力。它具有矢量算术运算和快速且节省空间的多维数组。

可以用标准数学函数对整组数据进行快速运算，不需要编写循环。具有用于读写磁盘数据的工具以及用于操作内存映射文件的工具。具有线性代数、随机数生成以及傅里叶变换功能。本节着重讲解利用 NumPy 进行数组操作。

NumPy 数据库不是 Python 的标准库，在使用之前，需要安装。安装 NumPy 的方法可以采用如下命令：

```
pip install numpy
```

NumPy 是 Python 的一个科学计算的库，提供了矩阵运算的功能，一般与 Scipy、Matplotlib 一起使用。其实，Python 中的 list 已经提供了类似于矩阵的表示形式，不过 NumPy 提供了更多的函数。如果接触过 MATLAB、Scilab，那么 NumPy 很好入手。在以下的代码示例中，总是先导入 NumPy：

```
import numpy as np
```

这里，as 保留字与 import 一起使用能够改变后续代码中库的命名空间，有助于提高代码可读性。简单地说，在程序的后续部分中，用 np 代替 NumPy，这样书写和阅读更简洁。

### 1. 一维数组

数组的类型是 numpy.ndarray。这一节使用 Python 命令行的形式讨论 NumPy 的数据处理功能。下面使用 numpy.array 方法以列表或元组变量为参数产生一维数组。>>>为命令行中输入的语句，其他行是控制台显示的内容。

```
>>> import numpy as np
>>>nList=np.array([1.0, 2.0, 3.0, 4.0])
>>>print(nList)
[1.  2.  3.  4.]
>>>print(type(nList))
<class 'numpy.ndarray'>
```

### 2. 二维数组

以列表或元组变量为元素产生二维数组：

```
>>>import numpy as np
>>>nList=np.array([[1.0, 2.0],[3.0, 4.0]])
>>>print(nList)
[[1.  2.]
 [3.  4.]]
```

生成数组时，可以指定数据类型，例如 numpy. int32、numpy. int16、numpy. float64 等：

```
>>>print(np.array([1.2,2,3,4], dtype=np.int32))
[1 2 3 4]
```

### 3. 序列数组

使用 numpy. arange 方法可以产生一个序列，对于一个生成的 numpy. ndarray 变量，还可以使用它自身的 reshape 方法变成多维数组。

```
>>>import numpy as np
>>>nList1=np.arange(15)
>>>print(nList1)
[ 0  1  2 ..., 12 13 14]
>>>nList2=np.arange(15).reshape(3,5)
>>>print(nList2)
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]
>>>print(type(nList1),type(nList2))
<class 'numpy.ndarray'><class 'numpy.ndarray'>
```

### 4. 等差序列数组

使用 numpy. linspace 方法可以生成具有一定间隔的数列。例如，在 1～3 中产生 9 个数：

```
>>>import numpy as np
>>>print(np.linspace(1, 3, 9))
[ 1.    1.25  1.5   1.75  2.    2.25  2.5   2.75  3.  ]
```

### 5. 特定的矩阵

使用 numpy. zeros、numpy. ones、numpy. eye 等方法可以构造特定的矩阵。zeros 生成零矩阵，ones 生成全为 1 的矩阵，eye 生成单位阵。

```
>>>print(np.zeros((3,4)))
[[ 0. 0. 0. 0.]
 [ 0. 0. 0. 0.]
 [ 0. 0. 0. 0.]]
>>>print(np.ones((3,4)))
[[ 1. 1. 1. 1.]
 [ 1. 1. 1. 1.]
 [ 1. 1. 1. 1.]]
>>>print np(eye(3))
[[ 1. 0. 0.]
 [ 0. 1. 0.]
 [ 0. 0. 1.]]
```

## 6. 获取数组的属性

```
>>>import numpy as np
>>>a=np.zeros((2,2,2))
>>>print(a.ndim)            #数组的维数
3
>>>print(a.shape)           #数组每一维的大小
(2, 2, 2)
>>>print(a.size)            #数组的元素数
8
>>>print(a.dtype)           #元素类型
float64
>>>print(a.itemsize)        #每个元素所占的字节数
8
```

## 7. 数组索引,切片和替换元素

```
>>>import numpy as np
>>>a=np.array( [[2,3,4],[5,6,7]] )
>>>print(a)
[[2 3 4]
[5 6 7]]
>>>print(a[1,2])            #访问第 2行第 3列的元素
7
>>>print(a[1,:])            #访问第 2行的所有元素
[5 6 7]
>>>print a[1,1:3]           #访问第 2行 [1,3)列的元素,即 1,2列
[6 7]
```

```
>>>a[1,:]=[8,9,10]          #替换第 2行的所有元素
>>>print a
[[ 2 3 4]
[ 8 9 10]]
```

## 8. 基本的数组运算

先构造数组 a、b,其中 a 是 2×2 的全 1 数组,b 是单位阵:

```
>>>import numpy as np
>>>a=np.ones((2,2))
>>>b=np.eye(2)
>>>print(a)
[[ 1. 1.]
[ 1. 1.]]
>>>print(b)
[[ 1. 0.]
[ 0. 1.]]
```

下面演示对数组的加减乘除运算:

```
>>>print(a>2)
[[False False]
[False False]]
>>>print(a+b)
[[ 2. 1.]
[ 1. 2.]]
>>>print(a-b)
[[ 0. 1.]
[ 1. 0.]]
>>>print(b*2)
[[ 2. 0.]
[ 0. 2.]]
>>>print((a*2)*(b*2))
[[ 4. 0.]
[ 0. 4.]]
>>>print(b/(a*2))
[[ 0.5 0. ]
[ 0. 0.5]]
>>>print((a*2)**4)                #**代表了指数运算
[[ 16. 16.]
[ 16. 16.]]
```

```
>>>print(a.sum())                 #计算元素的和
4.0
>>>print(b.min())                 #查找元素的最小值
0.0
>>>print(b.max())                 #查找元素的最大值
1.0
```

### 9. 基本的矩阵运算

矩阵的点乘：

```
>>>print(np.dot(a,b))
[[ 1.   1.]
[ 1.   1.]]
```

矩阵的叉乘：

```
>>>print(np.cross(a,b))
[-1.   1.]
```

矩阵的转置运算：

```
>>>a=np.array([[1,0],[2,3]])
>>>print(a)
[[1 0]
[2 3]]
>>>print(a.transpose())
[[1 2]
[0 3]]
```

计算矩阵的迹：

```
>>>a=np.array([[1,0],[2,3]])
>>>print(np.trace(a))
4
```

计算矩阵的特征值和特征向量。numpy. linalg 模块中有很多关于矩阵运算的方法，比如特征值和特征向量就是使用 numpy. linalg. eig 方法：

```
>>>import numpy as np
>>>import numpy.linalg as nplg
>>>a=np.array([[1, 0], [2, 3]])
```

```
>>>print(a)
[[1 0]
[2 3]]
>>>print(nplg.eig(a))
(array([ 3.,   1.]), array([[ 0.          ,  0.70710678],
       [ 1.          ,- 0.70710678]]))
```

## 6.3.2 Matplotlib 绘制图表

Matplotlib 是 Python 最著名的绘图库，它提供了一整套和 MATLAB 相似的命令 API，十分适合交互式地进行制图。而且也可以方便地将它作为绘图控件，嵌入 GUI 应用程序中。在官网上(http://matplotlib.org/)它的文档相当完备，并且 Gallery 页面中有上百幅缩略图，打开之后都有源程序。如果需要绘制某种类型的图，只需要在这个页面中浏览、复制、粘贴一下，就可以得到相应的源码。

安装 Matplotlib 一般采用如下的命令：

```
pip install matplotlib
```

Matplotlib 常用的函数如下。

### 1. figure()函数

作用新建绘画窗口，独立显示绘画的图表。调用 figure 函数创建一个绘图对象，并且使它成为当前的绘图对象。

```
plt.figure(figsize=(8,4))
```

figsize 参数指定绘图对象的宽度和高度，单位为英寸；dpi 参数指定绘图对象的分辨率，即每英寸多少个像素，默认值为 80。因此本例中所创建的图表窗口的宽度为 8×80＝640 像素。

### 2. plot()函数

这个函数比较常用，用于显示图形，它的用法为：

```
plt.plot(x,y,format_string,**kwargs)
```

参数的 x 为横轴数据，可为列表或数组。y 是纵轴数据，也是列表或者数组。format_string 为控制曲线的格式字符串，是可选项。**kwargs 为第二组数据或更多的(x，y，format_string)。format_string 控制曲线的格式，是由颜色字符、风格字符和标记字符组成。具体内容见表 6-4～表 6-6。

表 6-4　颜色字符

| 颜色字符 | 说　明 | 颜色字符 | 说　明 |
|---|---|---|---|
| 'b' | 蓝色 | 'm' | 洋红色 |
| 'g' | 绿色 | 'y' | 黄色 |
| 'r' | 红色 | 'k' | 黑色 |
| 'c' | 青绿色 | 'w' | 白色 |
| '#008000' | RGB 某颜色 | '0.8' | 灰度值字符串 |

表 6-5　风格字符

| 风格字符 | 说　明 | 风格字符 | 说　明 |
|---|---|---|---|
| '-' | 实线 | ':' | 虚线 |
| '--' | 破折线 | '' | 无线条 |
| '-.' | 点画线 | | |

表 6-6　标记字符

| 标记字符 | 说　明 | 标记字符 | 说　明 | 标记字符 | 说　明 |
|---|---|---|---|---|---|
| '.' | 点标记 | '1' | 下花三角标记 | 'h' | 竖六边形标记 |
| ',' | 像素标记(极小点) | '2' | 上花三角标记 | 'H' | 横六边形标记 |
| 'o' | 实心圈标记 | '3' | 左花三角标记 | '+' | 十字标记 |
| 'v' | 倒三角标记 | '4' | 右花三角标记 | 'x' | x 标记 |
| '^' | 上三角标记 | 's' | 实心方形标记 | 'D' | 菱形标记 |
| '>' | 右三角标记 | 'p' | 实心五角标记 | 'd' | 瘦菱形标记 |
| '<' | 左三角标记 | '*' | 星形标记 | '\|' | 垂直线标记 |

### 3．subplot()函数

Matplotlib 下，一个 Figure 对象可以包含多个子图，可以使用 subplot()函数快速绘制，主要作用是将多个图表绘在同一个窗口中。其调用形式如下：

```
subplot(numRows,numCols,plotNum)
```

图表的整个绘图区域被分成 numRows 行和 numCols 列。然后按照从左到右，从上到下的顺序对每个子区域进行编号，左上的子区域的编号为 1。plotNum 参数指定创建的图对象所在的区域。

如果 numRows＝2，numCols＝3，那整个绘制图表样式为 2×3 的区域，用坐标表示为：

(1,1),(1,2),(1,3)

(2,1),(2,2),(2,3)
这时,如果 plotNum=3,表示的坐标为(1,3),即第一行第三列的子图。

### 4. show()函数

这个函数是显示绘画的图表。
(1) 绘制基本的图表。
**【例 6-10】** 绘制简单的数据图表。

```
import matplotlib.pyplot as plt
from matplotlib import font_manager

zh_font= font_manager.FontProperties(fname= 'c:\Windows\Fonts\simhei.ttf', size= 14)
plt.figure(figsize= (10,9))
plt.plot([1, 2, 3])
plt.xlabel('横坐标说明', fontproperties= zh_font)
plt.ylabel('纵坐标说明', fontproperties= zh_font)
plt.show()
```

结果如图 6-4 所示。

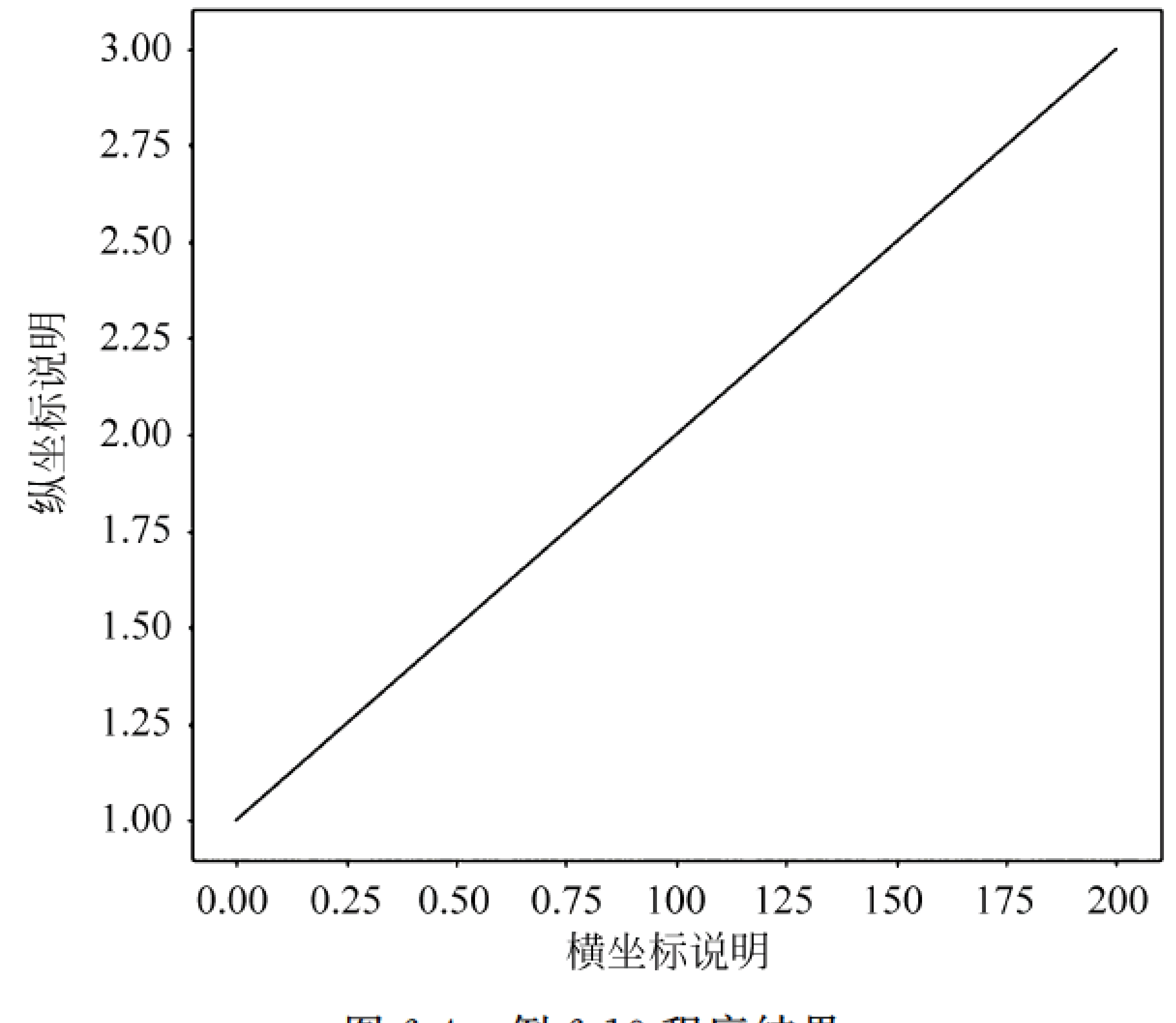


图 6-4 例 6-10 程序结果

从图 6-4 中可以看到,如果 plot()参数是一个列表(list)或数组(array),那么画图默认是作为 y 轴来显示,x 轴是自动生成的数值范围。其实 plot 可以带一些参数,如 plt.plot([1,2,3],[1,4,9]),则会按(1,1)、(2,4)、(3,9)来画线。当然,也可以指定线的类型和颜色,默认为"b—",即蓝色的实线。下面我们指定不同的颜色和类型。

**【例 6-11】** 指定红色圆点的图表。

```
import matplotlib.pyplot as plt

plt.figure(figsize= (10,9))
plt.plot([1,2,3,4], [1,4,9,16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```

结果如图 6-5 所示。

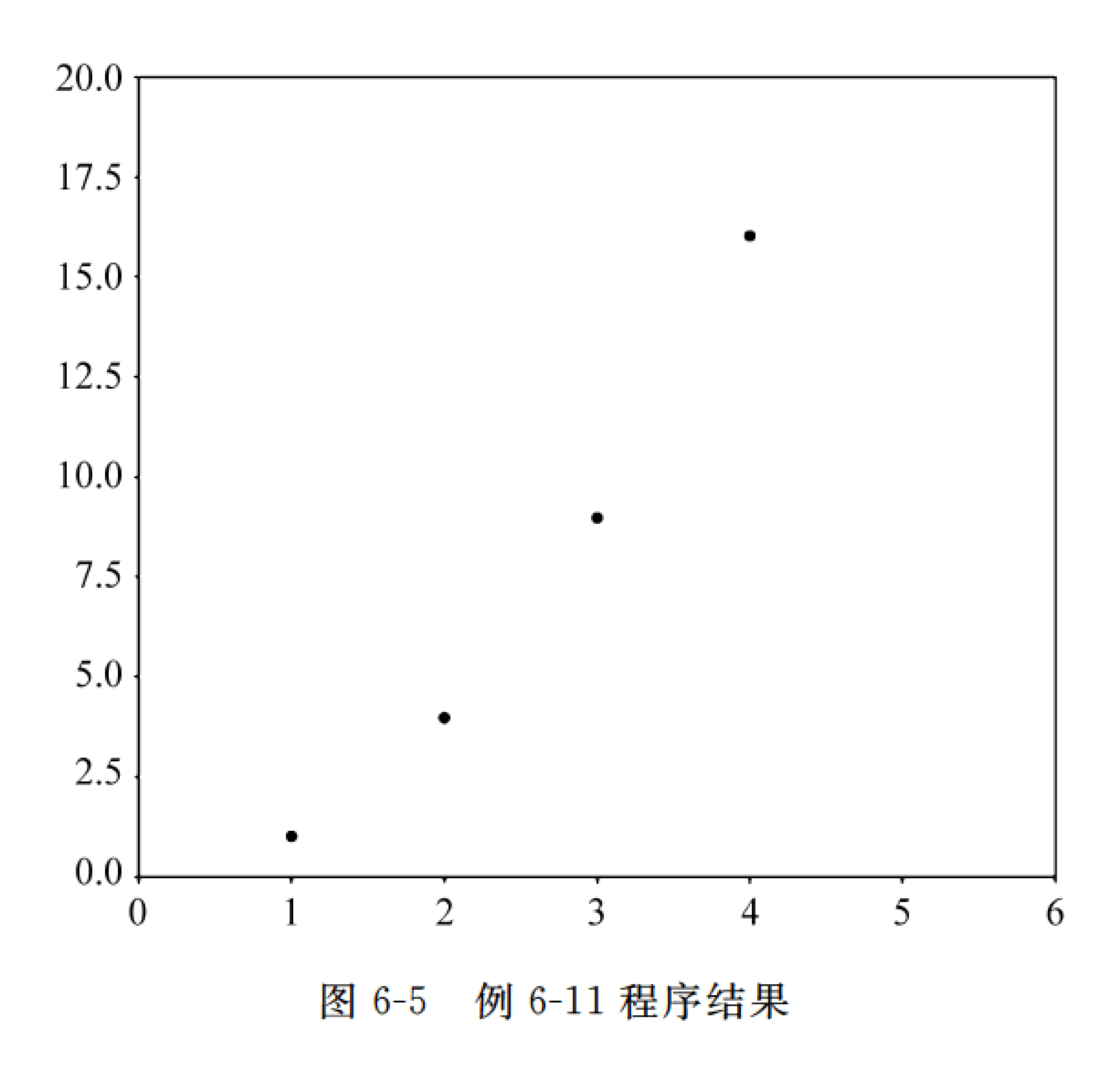



图 6-5　例 6-11 程序结果

'ro'代表线形为红色圈。plt.axis([0,6,0,20])是指定 x、y 坐标的起始范围，它的参数是列表[xmin,xmax,ymin,ymax]。

(2) 绘制子图。

Matplotlib 使用 subplot 命令绘制子图。

**【例 6-12】** 绘制子图。

```
import numpy as np
import matplotlib.pyplot as plt
def f(t):
    return np.exp(-t) * np.cos(2* np.pi * t)

t1=np.arange(0.0,5.0,0.1)
t2=np.arange(0.0,5.0,0.02)
plt.figure(1)
plt.subplot(211)
plt.plot(t1,f(t1),'bo',t2,f(t2),'k')
plt.subplot(212)
```

```
plt.plot(t2,np.cos(2* np.pi* t2),'r--')
plt.show()
```

结果如图 6-6 所示。

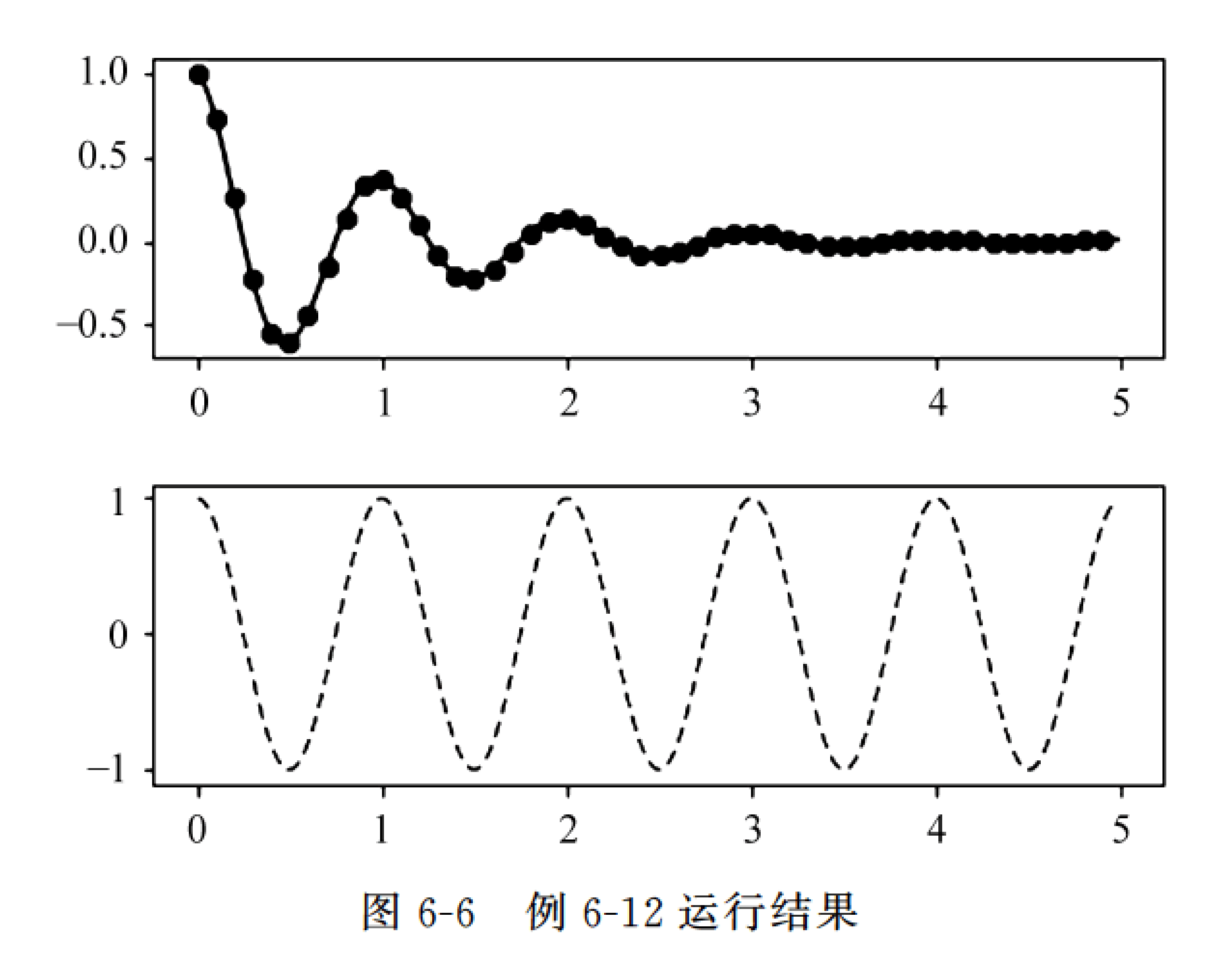



图 6-6 例 6-12 运行结果

**【例 6-13】** 使用 figure 绘制多图。

```
import matplotlib.pyplot as plt
#第一个窗口
plt.figure(1)
#第一个窗口中的第一个子图
plt.subplot(211)
plt.plot([1,2,3])
#第一个窗口中的第二个子图
plt.subplot(212)
plt.plot([4,5,6])
#第二个窗口
plt.figure(2)
#创建一个子图
plt.plot([4,5,6])
#figure 1 设为当前,subplot(212)还是当前图表 t
plt.figure(1)
#把 subplot(211)设为当前图表
plt.subplot(211)
#改变 subplot(211)的标题
plt.title('Easy as 1,2,3')
```

结果如图 6-7 所示。

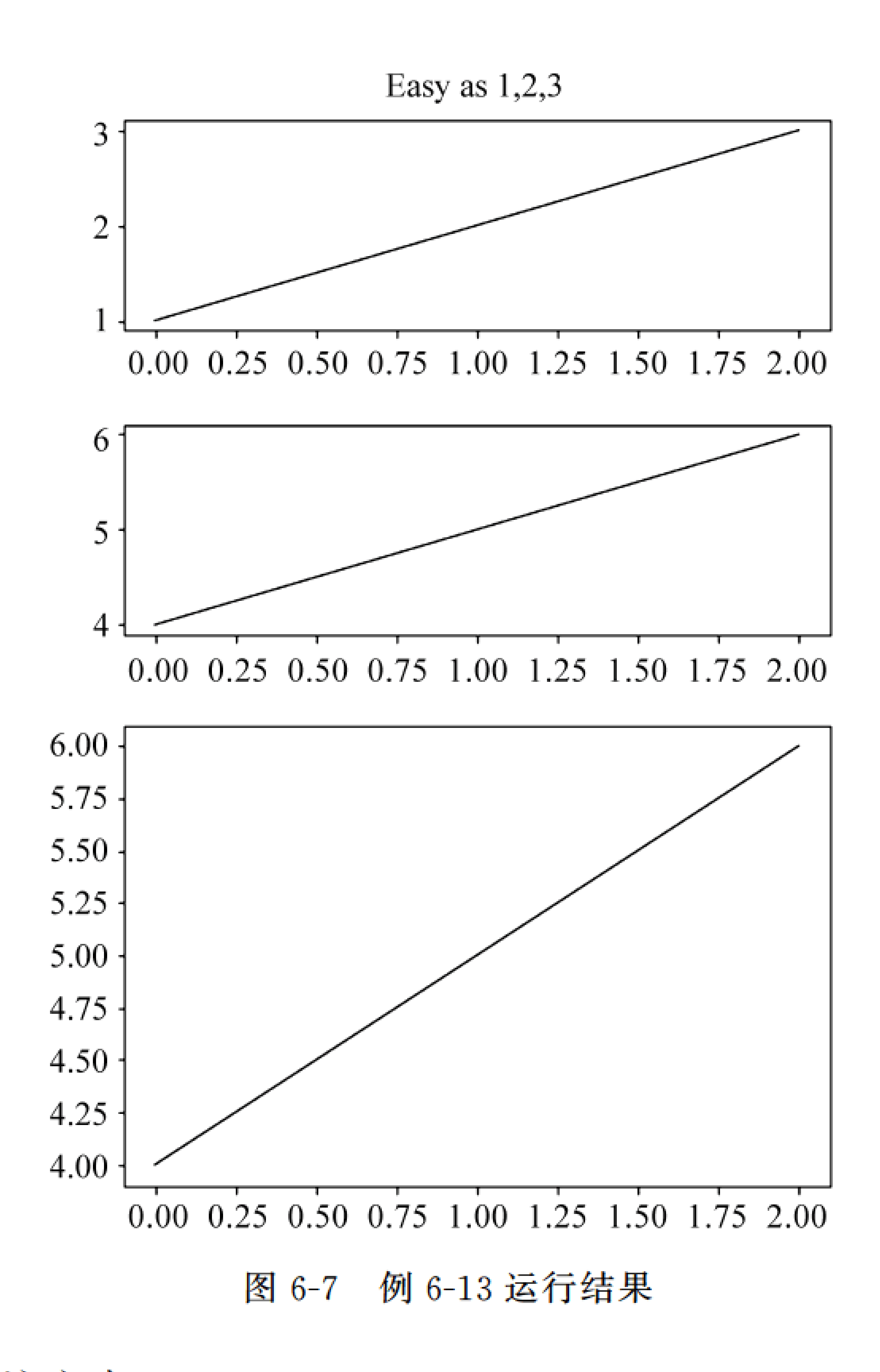



图 6-7　例 6-13 运行结果

（3）在图表上标注文本。

**【例 6-14】** 图表标注文字。

```
import numpy as np
import matplotlib.pyplot as plt

mu,sigma=100,15
x=mu+sigma*np.random.randn(10000)
#the histogram of the data
n,bins,patches=plt.hist(x,50,normed=1,facecolor='g',alpha=0.75)
plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title('Histogram of IQ')
plt.text(60,.025,r'$\mu=100,\ \sigma=15$')
plt.axis([40,160,0,0.03])
plt.grid(True)
plt.show()
```

结果如图 6-8 所示。

可以看到 Matplotlib 接受 Tex 的数学公式模式。借助 Latex，可以在图形中显示复杂的数学公式。

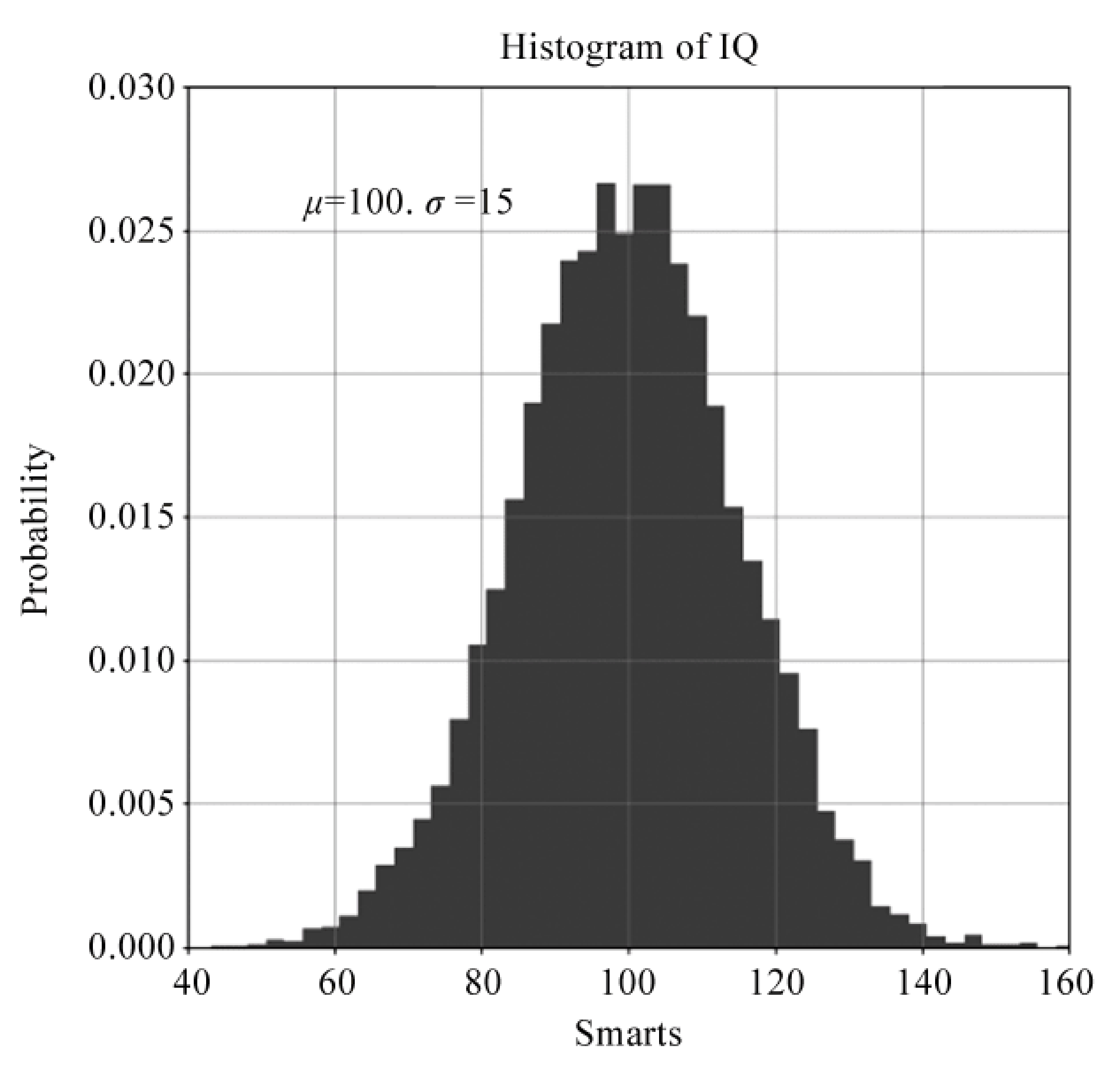



图 6-8　例 6-14 运行结果

## 6.3.3　SciPy 数值计算库

SciPy 函数库在 NumPy 库的基础上增加了众多的数学、科学以及工程计算中常用的库函数，例如线性代数、常微分方程数值求解、信号处理、图像处理、稀疏矩阵等。作为入门介绍，让我们看看如何用 SciPy 进行插值处理、信号滤波，以及用 C 语言加速计算。

### 1. 最小二乘拟合

假设有一组实验数据(x[i],y[i])，我们知道它们之间的函数关系：f=f(x)，通过这些已知信息，需要确定函数中的一些参数项。例如，如果 f 是一个线性函数 f(x)=kx+b，那么参数 k 和 b 就是我们需要确定的值。如果将这些参数用 p 表示的话，那么就是要找到一组 p 值使得如下公式中的 S 函数最小：

$$S(p) = \sum_{i=1}^{m}(y_i - f(x_i, p))^2$$

这种算法称为最小二乘拟合(Least-square fitting)。SciPy 中的子函数库 Optimize 提供了实现最小二乘拟合算法的函数 leastsq。下面是用 leastsq 进行数据拟合的一个例子。

**【例 6-15】**　数据拟合。

```
import numpy as np
from scipy.optimize import leastsq
```

```
import pylab as pl
from matplotlib import font_manager

def func(x, p):
    #数据拟合所用的函数:A* sin(2* pi* k* x+ theta)
    A, k, theta=p
    return A* np.sin(2* np.pi* k* x+ theta)
def residuals(p, y, x):
    #实验数据 x, y和拟合函数之间的差,p为拟合需要找到的系数
    return y- func(x, p)
x= np.linspace(0,- 2* np.pi, 100)
A, k, theta= 10, 0.34, np.pi/6          #真实数据的函数参数
y0= func(x, [A, k, theta])              #真实数据
y1= y0+ 2* np.random.randn(len(x))      #加入噪声之后的实验数据
p0= [7, 0.2, 0]                         #第一次猜测的函数拟合参数
#调用 leastsq进行数据拟合
#residuals 为计算误差的函数
#p0 为拟合参数的初始值
#args 为需要拟合的实验数据
plsq= leastsq(residuals, p0, args= (y1, x))
print("真实参数:", [A, k, theta])
print("拟合参数", plsq[0])
#实验数据拟合后的参数
zh_font= font_manager.FontProperties(fname= 'simfang.ttf', size= 14)
fig=pl.figure(figsize= (18,10))
pl.plot(x, y0, label= "真实数据")
pl.plot(x, y1, label= "带噪声的实验数据")
pl.plot(x, func(x, plsq[0]), label= "拟合数据")
pl.legend(prop= zh_font)
pl.show()
```

这个例子中要拟合的函数是一个正弦波函数,它有三个参数 A、k、theta,分别对应振幅、频率、相角。假设我们的实验数据是一组包含噪声的数据 x、y1,其中 y1 是在真实数据 y0 的基础上加入噪声的。

通过 leastsq 函数对带噪声的实验数据 x、y1 进行数据拟合,可以找到 x 和真实数据 y0 之间的正弦关系的三个参数:A、k、theta。程序输出如图 6-9 所示。

```
真实参数: [10,0.34,0.5235987755982988]
拟合参数 [- 9.81677284   0.33933551   3.69031748]
```

我们看到拟合参数虽然和真实参数完全不同,但是由于正弦函数具有周期性,实际上拟合参数得到的函数和真实参数对应的函数是一致的。

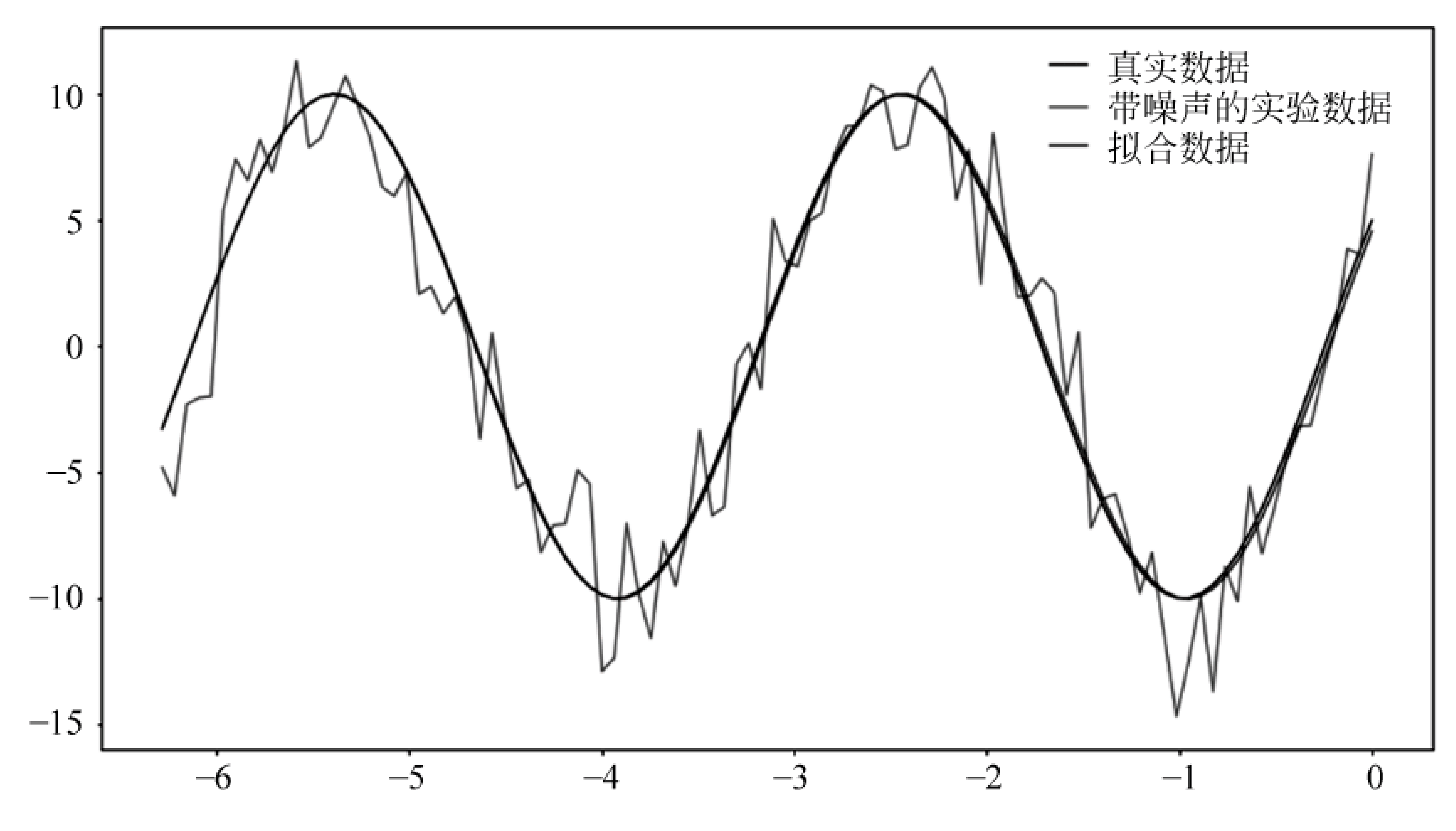



图 6-9　调用 leastsq 函数对噪声正弦波数据进行曲线拟合

## 2. 函数最小值

Optimize 库提供了几个求函数最小值的算法：fmin、fmin_powell、fmin_cg、fmin_bfgs。下面的程序通过求解卷积的逆运算演示 fmin 的功能。

对于一个离散的线性时不变系统 h，如果它的输入是 x，那么其输出 y 可以用 x 和 h 的卷积表示：

$$y = x * h$$

现在的问题是，如果已知系统的输入 x 和输出 y，如何计算系统的传递函数 h；或者如果已知系统的传递函数 h 和系统的输出 y，如何计算系统的输入 x。这种运算被称为反卷积运算，计算是十分困难的，特别是在实际的运用中，测量系统的输出总是存在误差的。

下面用 fmin 计算反卷积，这种方法只能用在很小规模的数列之上，因此没有很大的实用价值，不过用来评价 fmin 函数的性能还是不错的。

**【例 6-16】** 求函数最小值。

```
#本程序用各种 fmin 函数求卷积的逆运算
import scipy.optimize as opt
import numpy as np
def test_fmin_convolve(fminfunc, x, h, y, yn, x0):
    #x (* ) h=y, (* )表示卷积
    #yn 为在 y的基础上添加一些干扰噪声的结果
    #x0 为求解 x的初始值
def  convolve_func(h):
        #计算 yn-x (* ) h的 power
        #fmin将通过计算使得此 power最小
        return np.sum((yn-np.convolve(x, h))**2)

h0=fminfunc(convolve_func, x0)
```

```
        #调用 fmin 函数,以 x0 为初始值
print(fminfunc.__name__)
print("---------------------")
    #输出 x(*)h0 和 y 之间的相对误差
print("error of y:", np.sum((np.convolve(x, h0)-y)**2)/np.sum(y**2))
    #输出 h0 和 h 之间的相对误差
print("error of h:", np.sum((h0-h)**2)/np.sum(h**2))
print()
def test_n(m, n, nscale):
    #随机产生 x, h, y, yn, x0 等数列,调用各种 fmin 函数求解 b
    #m 为 x 的长度, n 为 h 的长度, nscale 为干扰的强度
    x=np.random.rand(m)
    h=np.random.rand(n)
    y=np.convolve(x, h)
    yn=y+np.random.rand(len(y))*nscale
    x0=np.random.rand(n)

    test_fmin_convolve(opt.fmin, x, h, y, yn, x0)
    test_fmin_convolve(opt.fmin_powell, x, h, y, yn, x0)
    test_fmin_convolve(opt.fmin_cg, x, h, y, yn, x0)
    test_fmin_convolve(opt.fmin_bfgs, x, h, y, yn, x0)
    if __name__=="__main__":
    test_n(200, 20, 0.1)
```

程序的输出结果如下：

```
fmin
---------------------
error of y: 0.00215423136494
error of h: 0.0940355093353

Optimization terminated successfully.
         Current function value: 0.207532
         Iterations: 41
         Function evaluations: 7644
fmin_powell
---------------------
error of y: 0.000141560690383
error of h: 0.000212378358534

Optimization terminated successfully.
         Current function value: 0.207529
         Iterations: 24
```

```
          Function evaluations: 1100
          Gradient evaluations: 50
fmin_cg
----------------------
error of y: 0.000141659918392
error of h: 0.000213318921195

Optimization terminated successfully.
          Current function value: 0.207529
          Iterations: 31
          Function evaluations: 946
          Gradient evaluations: 43
fmin_bfgs
----------------------
error of y: 0.00014165992857
error of h: 0.000213319218804
```

### 3. 非线性方程组求解

Optimize 库中的 fsolve 函数可以用来对非线性方程组进行求解。它的基本调用形式如下：

```
fsolve(func,x0)
```

func(x)是计算方程组误差的函数，它的参数 x 是一个矢量，表示方程组的各个未知数的一组可能解，func 返回将 x 代入方程组之后得到的误差；x0 为未知数矢量的初始值。如果要对如下方程组进行求解：

```
f1(u1,u2,u3)=0
f2(u1,u2,u3)=0
f3(u1,u2,u3)=0
```

那么 func 可以如下定义：

```
def func(x):
    u1,u2,u3=x
    return [f1(u1,u2,u3), f2(u1,u2,u3), f3(u1,u2,u3)]
```

下面是一个实际的例子，求解如下方程组的解：

$$5 * x_1 + 3 = 0$$

$$4 * x_0 * x_0 - 2 * \sin(x_1 * x_2) = 0$$

$$x_1 * x_2 - 1.5 = 0$$

【例 6-17】 线性方程组的解法。

```
from scipy.optimize import fsolve
from math import sin,cos
def f(x):
    x0=float(x[0])
    x1=float(x[1])
    x2=float(x[2])
    return [
        5*x1+3,
        4*x0*x0-2*sin(x1*x2),
        x1*x2-1.5
    ]
result=fsolve(f, [1,1,1])
print(result)
print(f(result))
```

输出结果为：

```
[-0.70622057    -0.6           -2.5          ]
[0.0, -9.126033262418786 8e-14, 5.329070518200751 4e-15]
```

由于 fsolve 函数在调用函数 f 时，传递的参数为数组，因此如果直接使用数组中的元素计算，计算速度将会有所降低，所以这里先用 float 函数将数组中的元素转换为 Python 中的标准浮点数，然后调用标准 math 库中的函数进行运算。

在对方程组进行求解时，fsolve 会自动计算方程组的雅可比矩阵，如果方程组中的未知数很多，而与每个方程有关的未知数较少时，即雅可比矩阵比较稀疏时，传递一个计算雅可比矩阵的函数可以大幅度提高运算速度。笔者在一个模拟计算的程序中需要大量求解近有 50 个未知数的非线性方程组的解。每个方程平均与 6 个未知数相关，通过传递雅可比矩阵的计算函数使计算速度提高了 4 倍。

### 4. 雅可比矩阵

雅可比矩阵是一阶偏导数以一定方式排列的矩阵，它给出了可微分方程与给定点的最优线性逼近，因此类似于多元函数的导数。例如，前面的函数 f1、f2、f3 和未知数 u1、u2、u3 的雅可比矩阵如下：

$$\begin{bmatrix} \frac{\partial f1}{\partial u1} & \frac{\partial f1}{\partial u2} & \frac{\partial f1}{\partial u3} \\ \frac{\partial f2}{\partial u1} & \frac{\partial f2}{\partial u2} & \frac{\partial f2}{\partial u3} \\ \frac{\partial f3}{\partial u1} & \frac{\partial f3}{\partial u2} & \frac{\partial f3}{\partial u3} \end{bmatrix}$$

使用雅可比矩阵的 fsolve 示例如下，计算雅可比矩阵的函数 j 通过 fprime 参数传递

给 fsolve，函数 j 和函数 f 一样，有一个未知数的解矢量参数 x，函数 j 计算非线性方程组在矢量 x 点上的雅可比矩阵。

**【例 6-18】** 求雅可比矩阵。

```
from scipy.optimize import fsolve
from math import sin,cos
def f(x):
    x0=float(x[0])
    x1=float(x[1])
    x2=float(x[2])
    return [
        5*x1+3,
        4*x0*x0-2*sin(x1*x2),
        x1*x2-1.5
    ]
def j(x):
    x0=float(x[0])
    x1=float(x[1])
    x2=float(x[2])
    return [
        [0, 5, 0],
        [8*x0,-2*x2*cos(x1*x2),-2*x1*cos(x1*x2)],
        [0, x2, x1]
    ]
result=fsolve(f, [1,1,1], fprime=j)
print(result)
print(f(result))
```

运行结果为：

```
[-0.70622057 -0.6          -2.5         ]
[0.0,-9.126033262418787e-14, 5.329070518200751e-15]
```

## 5. B 样条曲线

interpolate 库提供了许多对数据进行插值运算的函数。下面是使用直线和 B 样条对正弦波上的点进行插值的例子。

**【例 6-19】** B 样条插值。

```
import numpy as np
import pylab as pl
from scipy import interpolate
```

```
from matplotlib import font_manager

zh_font=font_manager.FontProperties(fname='simfang.ttf', size=18)
x=np.linspace(0, 2*np.pi+np.pi/4, 10)
y=np.sin(x)
x_new=np.linspace(0, 2*np.pi+np.pi/4, 100)
f_linear=interpolate.interp1d(x, y)
tck=interpolate.splrep(x, y)
y_bspline=interpolate.splev(x_new, tck)
fig=pl.figure(figsize=(18,10))
pl.plot(x, y, "o", label="原始数据")
pl.plot(x_new, f_linear(x_new), label="线性插值")
pl.plot(x_new, y_bspline, label="B-spline 插值")
pl.legend(prop=zh_font)
pl.show()
```

在这段程序中，通过 interp1d 函数直接得到一个新的线性插值函数。而 B 样条插值运算需要先使用 splrep 函数计算出 B 样条曲线的参数，然后将参数传递给 splev 函数计算出各个取样点的插值结果，如图 6-10 所示。

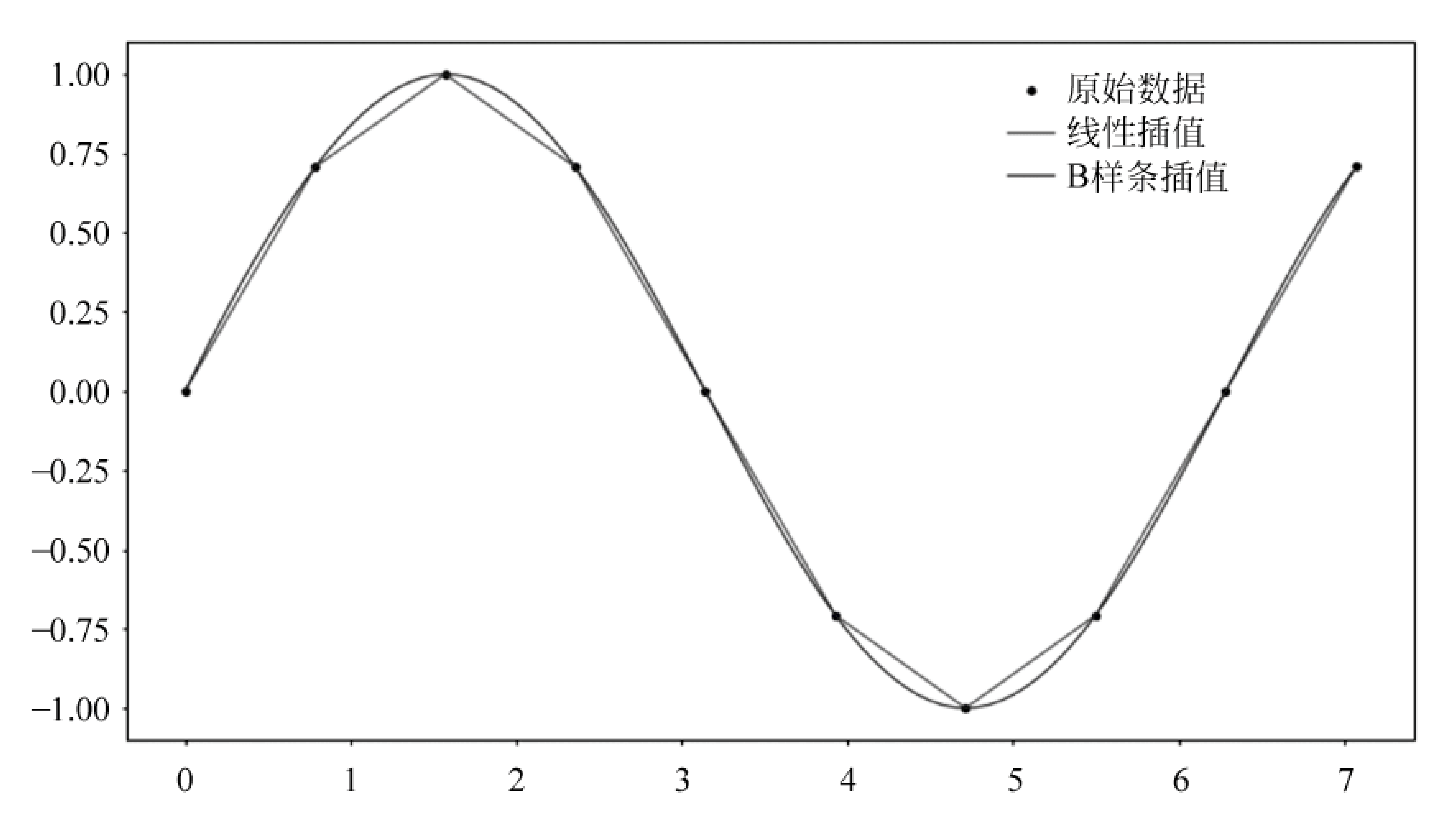



图 6-10　使用 interpolate 库对正弦波数据进行线性插值和 B 样条插值

## 6. 数值积分

数值积分是对定积分的数值求解，例如可以利用数值积分计算某个形状的面积。下面让我们来考虑一下如何计算半径为 1 的半圆面积，根据圆的面积公式，其面积应该等于 PI/2。单位半圆曲线可以用下面的函数表示：

```
def half_circle(x):
    return (1-x**2)**0.5
```

下面程序使用经典的分小矩形来计算面积总和的方式，计算出单位半圆的面积。

【例 6-20】 利用小面积之和计算圆面积。

```
import numpy as np

def half_circle(x):
    return (1-x**2)**0.5
N=10000
x=np.linspace(-1, 1, N)
dx=2.0/N
y=half_circle(x)
m=np.sum(y*dx)*2          #面积的两倍
print(m)
```

得到的圆面积是 3.141275168。利用上述方式计算出圆上一系列点的坐标，还可以用 numpy.trapz 进行数值积分。

【例 6-21】 利用 numpy.trapz 进行数值积分。

```
import numpy as np

def half_circle(x):
    return (1-x**2)**0.5
N=10000
x=np.linspace(-1, 1, N)
dx=2.0/N
y=half_circle(x)
m=np.trapz(y, x)*2                    #面积的两倍
print(m)
```

得到的圆面积是 3.14158932693。此函数计算的是以 x，y 为顶点坐标的折线与 X 轴所夹的面积。同样的分割点数，trapz 函数的结果更加接近精确值一些。

如果我们调用 scipy.integrate 库中的 quad 函数的话，将会得到非常精确的结果。

【例 6-22】 利用 quad 函数求积分。

```
import numpy as np
from scipy import integrate

def half_circle(x):
    return (1-x**2)**0.5
N=10000
x=np.linspace(-1, 1, N)
dx=2.0/N
```

```
y=half_circle(x)
m, err=integrate.quad(half_circle,-1, 1)
print(m* 2)
```

得到的圆面积是 3.141592653589797。

## 本章小结

(1) 掌握 Python 实现类和对象的基本方法。
(2) 理解 turtle 函数的设计图形方法。
(3) 掌握 Numpy 第三方模块库的基本函数。
(4) 掌握 Scipy 第三方模块库的基本函数。
(5) 掌握 Matplotlib 第三方模块库的基本函数。
(6) 了解利用 Python 标准函数解决基本数学问题。

## 习　　题

### 1. 简答题

(1) 简述什么是模块、如何把模块导入解释器,以及几种导入方法。
(2) 在解释器中如何终止程序并返回消息?

### 2. 填空题

(1) 面向对象程序设计的三要素分别为________、________、________。
(2) 通过________语句可以导入模块。
(3) 若一模块名为 m,________可以导入模块中的所有成员。
(4) 在 Python 中,每个模块都会有自己的名称,可以通过特殊变量________查看;特别地,当一个模块被单独运行时,模块名称为________。

### 3. 选择题

(1) 同一类型的不同实例之间不具备【　　】。
A) 相同的操作集合　　B) 相同的属性集合
C) 相同的对象名　　D) 不同的对象名
(2) 下列选项中,【　　】不是 OOP 的基本特征。
A) 类属型　　B) 继承　　C) 封装　　D) 多态

(3) 下列程序段返回的结果是【　　】。

```
class university:
    name= "BJTU"
    age= 120
p= university()
print(p.name)
```

A) BJTU　　B) "BJTU"　　C) 120　　D) SyntaxError

(4) 下列说法错误的是【　　】。

A) def 是定义方法的关键字

B) class 是定义类的关键字

C) 类是对现实世界中一些事物的封装

D) 方法是对现实世界中一些事物的封装

(5) 描述对象静态特征的数据元素是【　　】。

A) 方法　　B) 类型　　C) 属性　　D) 消息

(6) 在 Python 的类定义中,对成员变量的访问形式为【　　】。

A) <对象>.<变量>　　B) <类名>.<变量>

C) <对象>.方法(变量)　　D) <类名>.方法(变量)

(7) 下列选项中【　　】不是面向对象方法的优点。

A) 符合人们习惯的思维方法　　B) 以功能分析为中心

C) 代码复用率高　　D) 更容易维护

(8) 当一个类定义了【　　】方法后,类实例化时会自动调用该方法。

A) auto()　　B) init()

C) __auto__()　　D) __init__()

(9) 有子类 China 和 Japan 继承了父类 Asia,若 c 和 j 分别是以上两个子类的实例,则 isinstance(c,Asia)、isinstance(j,Asia)、isinstance(c,China)、isinstance(j,China)返回的结果分别是【　　】。

A) True True False False　　B) True True True False

C) True False False True　　D) True True True True

(10) 面向对象程序设计着重于【　　】的设计。

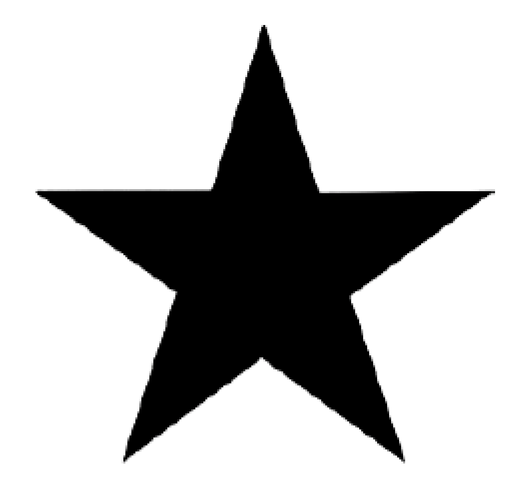
A) 对象　　B) 类　　C) 算法　　D) 数据

(11) 面向对象程序设计中,把对象的属性和行为组织在同一个模块内的机制叫做【　　】。

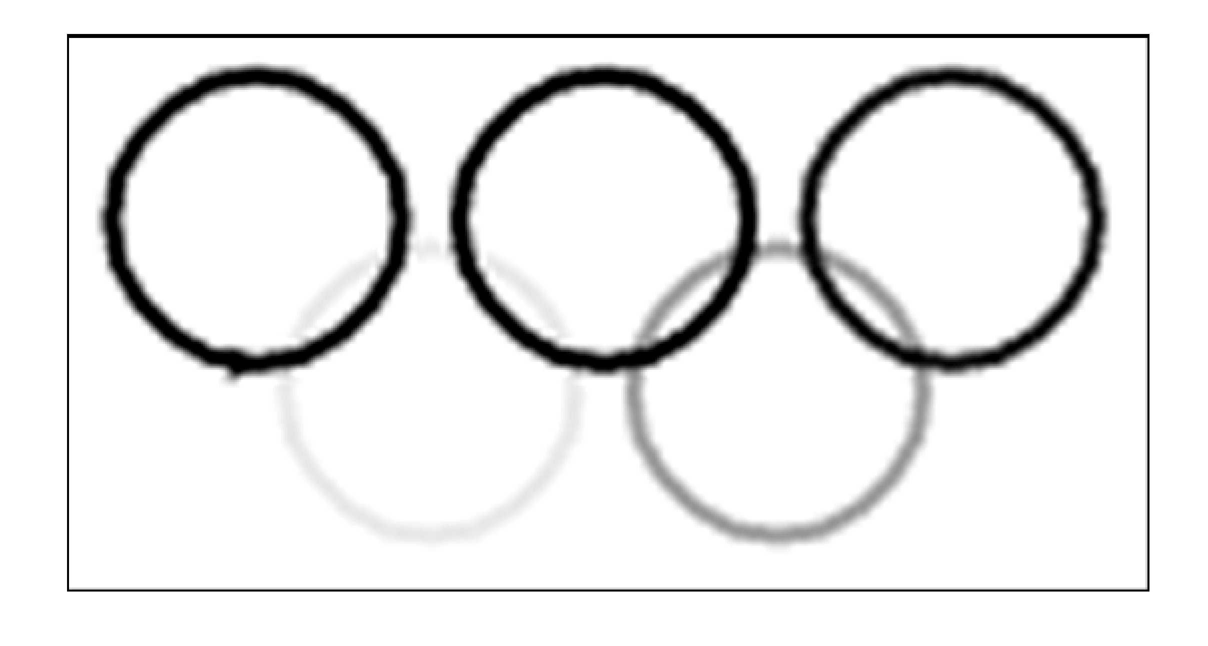
A) 抽象　　B) 继承　　C) 封装　　D) 多态

## 4. 程序设计题

(1) 画一个五角星,如下所示,注意填充为红色。



(2) 绘制奥运五环图，其中五种颜色分别为蓝色、黑色、红色、黄色和绿色。注意根据实际效果调整圆形的大小和位置。



# 第 7 章　综合训练题

1. 在一个壶里装有 141 个球，其中有 83 个白球和 58 个红球。从壶中随机取球两次，分别考虑两种取球方式：放回和不放回。求取到两个都是白球的概率和取到两个球中至少有一个是白球的概率。

```
from random import sample
import numpy as np
urn= ________________
sims= 10000
success= np.zeros(sims)
for i in range(sims):
    draw= sample(urn,2)
if sum(draw)==2:
    success[i]= sum(draw)
print("取到两个都是白球的概率是{}".format(round(np.mean(success),1)))
```

(1) 请补全语句"urn=________"。

(2) sims 的作用是什么？ sims＝10000 和 sims＝1000000 有什么区别？会有什么影响？

(3) success 的作用是什么？尝试使用另外的方法代替 np.zeros()。

(4) 语句 draw＝sample(urn,2)的作用是什么？draw 的类型是什么？请列举可能的结果。

(5) 请描述 sum(draw)＝＝2 在数学上的含义。

(6) 若 sum(draw)等于 2，success 的值会有什么变化？若不等于 2，success 的值会改变吗？

(7) sum(draw)语句 success[i]＝sum(draw)是否正确？如果错误，请修改程序。

(8) 为什么不直接输出 np.mean(success)？ round(np.mean(success),1)的作用是什么？round()的第二个参数有什么作用？为什么不直接在{}内控制格式，比如{:.1f}？

(9) 上述程序求出的是取到两个都是白球的概率还是取到两个球中至少有一个是白球的概率？请编程求出另一个概率。

(10) 请问上述程序的取球方式是什么？请继续求出另外一种取球方式的两个概率。

2. 某运输公司有 753 辆车参加保险，在一年内汽车出事故的概率是 0.006，每辆车每年交的保险费为 800 元，且最高赔偿 50000 元，请计算保险公司一年获利不小于 350000

元的概率。

（1）stats. binom. pmf 中三个模块名分别具有什么含义？

（2）分别解释 range(6)、753、0. 006 的含义。

（3）对比上一题，为什么使用了 sum()？ sum()接收了什么参数？ stats. binom. pmf()的返回值是什么？

（4）尝试去掉 float，看看输出结果有什么不同。对比上一题，为什么使用了 float()？sum()的返回值是什么？

```
from scipy import stats
print(round(float(sum(stats.binom.pmf(range(6),753,0.006))),1))
'''
print(
round(
float(
sum(
stats.binom.pmf(range(6),753,0.006)
)
)
,1)
)
'''
```

# 模 拟 题 一

院系：__________ 专业：__________ 班级：______

姓名：________ 学号：________

| 题号 | 一 | 二 | 三 | 四 | 总分 |
|---|---|---|---|---|---|
| 得分 | | | | | |

## 一、填空题(每题 1 分,共 10 分)

1. Python 语言有两种注释方法,分别是________。
2. s="hello",t="world",则 s[2:8]的值为________。
3. 表达式 30−3**2+8//3**2 * 10 的值为________。
4. s='Python String',则 s.upper()值是________。
5. 从 random 库中选取相应函数,随机生成 100 以内的奇数,应使用的函数是________。
6. 格式化输出 0.002178 对应的科学表示法形式,保留小数点后 3 位有效位的标准浮点形式以及百分形式为________。
7. Python 中________可以理解为对一组表达特定功能表达式的封装。
8. range(1,10,3)的值是________。
9. 跳出 for 循环,但仍然继续执行 for 循环外的语句是________。
10. 请分析下面的程序,若输入 score 为 80,输出 grade 为________。

```
if  score>=60.0:
     grade='D'
elif  score>=70.0:
      grade='C'
elif  score>=80.0:
      grade='B'
elif  score>=90.0:
      grade='A'
print(grade)
```

## 二、选择题(每题 2 分,共 20 分)

1. 下面属于流程图的基本元素的是【　　】。
   A) 判断框　　B) 顺序结构　　C) 分支结构　　D) 循环结构
2. 以下不属于 Python 合法变量名的是【　　】。

A) var-name　　B) !@#$%　　C) 1_elif　　D) sale_2008

3. 判断操作是否在分支结构中的依据是【　　】。

A) 括号　　B) 缩进　　C) 花括号　　D) 冒号

4. 以下为不合法布尔表达式的是【　　】。

A) x in range(6)　　B) 3=a

C) e>5 and 4==f　　D) 'abc' > 'xyz'

5. 以下不为 while 循环特点的是【　　】。

A) 提高程序的复用性　　B) 能够实现无限循环

C) 如果不小心会出现死循环　　D) 必须提供循环的次数

6. 正确解释以下语句执行结果的是【　　】。

```
>>>print(1.2-1.0==0.2)
    False
```

A) Python 的实现有错误　　B) 浮点数无法精确表示

C) 布尔运算不能用于浮点数比较　　D) Python 将非 0 数视为 False

7. 给定 list 为整数列表,以下程序的目的是【　　】。

```
def  function(list):
    for i in range(len(list)-1):
        if(abs(list[i]-list[i+1])>1):
            return False
    return True
```

A) 总是返回 True

B) 如果列表的长度为偶数,则返回 True

C) 如果列表中的数字按升序排列,则返回 True

D) 如果相邻数字之间的差值不超过 1,则返回 True

8. 下列语句在 Python 中非法的是【　　】。

A) x=y=z=1　　B) x=(y=z+1)　　C) x,y=y,x　　D) x+=y

9. 下述 while 循环执行的次数为【　　】。

```
k=1000
while k>1:
    print(k)
    k=k/2
```

A) 9　　B) 10　　C) 11　　D) 1000

10. 执行下列语句后的显示结果是【　　】。

```
>>>from math import sqrt
>>>print(sqrt(3)*sqrt(3)==3)
```

A) 3　　B) True

C) false　　D) sqrt(3)*sqrt(3)==3

## 三、阅读程序题(每题 6 分,共 30 分)

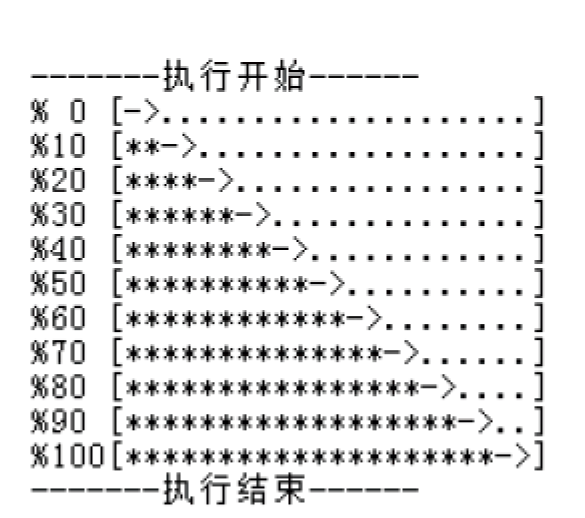
1. 下面程序中 a 为非负数,b 是正数,则该程序实现什么功能?写出 examCode(23,5)的执行结果。

```
def  examCode(a, b):
    val=a
    while val>b:
        val-=b
    return val
```

2. 写出下面程序的执行结果:

```
def  func(n):
    if  n==1:
        return 1
    if  n==2:
        return 2
    return  func(n-1)+func(n-2)
    print(func(5))
```

3. 下面程序完成进度条输出功能,输出效果如下图,请把程序补充完整。


```
-------执行开始------
% 0 [->....................]
%10 [**->..................]
%20 [****->................]
%30 [******->..............]
%40 [********->............]
%50 [**********->..........]
%60 [************->........]
%70 [**************->......]
%80 [****************->....]
%90 [******************->..]
%100[********************->]
-------执行结束------
```

```
import time
scale=10
print("-------执行开始------")
fori in range():
    a='**'*i
    b='..'*(scale-i)
    c=(i/scale)*100
print("".format(c,a,b))
time.sleep(0.1)
```

______________________

______________________

______________________

______________________

______________________

______________________

4. 下面程序要求用户输入二进制数字 0 或 1 并显示，请找出程序中的错误并改正。

```
bit= input("Enter a binary digit:")
if bit= 0 or 1:
    print("Your input is:{}",bit)
else
    print("Your input is invalid.")
```

5. 下面程序实现什么功能？写出程序执行结果。

```
a= [1, 20, 32, 14, 5, 62, 78, 38, 9, 10]
for i in range(9):
    if(a[i]> a[i+ 1]):
        a[i], a[i+ 1]= a[i+ 1], a[i]
print(a)
```

## 四、程序设计(每题 10 分，共 40 分)

1. 编写一个函数(流程图和 python 程序)：输入三个数，并输出其最大者。(5 分)

2. 编写一个函数，计算一个给定的日期是该年的第几天。(10 分)

3. 用 Python 编程，假设一年定期利率为 3.25%，计算一下需要过多少年，一万元的一年定期存款连本带息能翻番？(10 分)

4. 编写程序，企业发放的奖金按利润提成。利润(I)提成方式为：

① 低于或等于 10 万元时，奖金按 10%提成；

② 高于 10 万元但低于 20 万元时，低于 10 万元的部分按 10%提成，高于 10 万元的部分，按 7.5%提成；

③ 20 万～40 万时，高于 20 万元的部分，按 5%提成；

④ 40 万～60 万时，高于 40 万元的部分，按 3%提成；

⑤ 60 万～100 万时，高于 60 万元的部分，按 1.5%提成；

⑥ 高于 100 万元时，超过 100 万元的部分，按 1%提成。

从键盘输入当月利润 I，求应发放奖金总数。(15 分)

# 模 拟 题 二

院系：__________ 专业：__________ 班级：______

姓名：________ 学号：________

| 题号 | 一 | 二 | 三 | 四 | 总分 |
|---|---|---|---|---|---|
| 得分 | | | | | |

## 一、填空题(每题 2 分,共 20 分)

1. 设 s="Python Programming",那么 print(s[-5:])的结果是【　　】。

A) mming　　B) Pytho　　C) mmin　　D) Python

2. 以下程序的输出结果是(提示：ord('a')==97)【　　】。

```
lista=[1,2,3,4,5,'a','b','c','d','e']
print (lista[2]+lista[5])
```

A) 100　　B) 'd'　　C) d　　D) TypeError

3. 以下 bool 表达式执行结果为 True 的是【　　】。

A) 'pyhton'<'Python'　　B) 'BJTU'<'BJTU1'

C) 2<0　　D) 5<4 and 3>6

4. 执行下列语句后的显示结果是【　　】。

```
>>>a=1
>>>b=2* a / 4
>>>a="one"
>>>print(a,b)
```

A) one 0　　B) 1 0　　C) one 0)5　　D. one,0.5

5. 关于字符串下列说法错误的是【　　】。

A) 字符应该视为长度为 1 的字符串

B) 字符串以\0 标志字符串的结束

C) 既可以用单引号,也可以用双引号创建字符串

D) 在三引号字符串中可以包含换行回车等特殊字符

6. 下列不是正确字典创建方式的是【　　】。

A) d={1:[1,2],3:[3,4]}　　B) d={(1,2):1,(3,4):3}

C) d={1:"张三",2:"李四"}　　D) d={1;"张三",2;"李四"}

7. 以下可以终结一个循环的保留字是【　　】。

A) if　　B) break　　C) exit　　D) continue

8. 以下属于不合法的变量名是【　　】。

A）Bjtu_2017　　B）520bjtu　　C）_hello　　D）x

9. 假设文件 1.txt 包含 3 行数据，下列代码没有错误的是【　　】。

A）fr=open("1.txt", 'r')
　　line=fr.readline()
　　"1.txt".close()

B）fr=open("1.txt", 'w')
　　line=fr.readline()
　　fr.close()

C）fr=open("1.txt", 'r')
　　line=fr.write("D\n")
　　fr.close()

D）fr=open("1.txt", 'a')
　　line=fr.write("D\n")
　　fr.close()

10. 关于 returnr 的下列说法，正确的是【　　】。

A）如果函数中没有 return 语句，则默认返回空值 None

B）如果在函数中有语句 return 3，那么该函数一定会返回整数 3

C）函数中必须包含 return 语句

D）函数中的 return 语句一定能够得到执行

## 二、填空题（每题 1 分，共 10 分）

1. 文件类型有两种，分别是________。
2. 已知 x={'a':'b','c':'d'}，则表达式 'b' in x.values() 的值为________。
3. 假设列表对象 aList 的值为[3,4,5,6,7,9,11,13,15,17]，那么 aList[3:7]得到的值是________。
4. '{}.{}.{}'.format('www','pythontab','com') 的输出结果是________。
5. Python 语句 list(range(1,10,3))的执行结果为________。
6. 表达式 2*2**3%5 的值为________。
7. Python 中用于表示逻辑与、逻辑或、逻辑非运算的关键字分别是________。
8. 已知 a='hello',b='Python'，则表达式 a[3:]+b[:-3]的值为________。
9. 表达式 [1,2]*2 的值为________。
10. 列表、元组、字符串是 Python 的________序列。

## 三、阅读程序题（每题 6 分，共 30 分）

1. 解释程序完成的主要功能，并给出 n=5 的结果。

```
n=int(input("Please enter a whole number: "))
fact=1
for factor in range(n,1,-1):
    fact=fact*factor
print("The factorial of", n, "is", fact)
```

________________________________________

2. 写出下面程序的执行结果。

```
def Sum(a, b=3, c=5):
```

```
return sum([a, b, c])
print(Sum(a=8, c=2))
print(Sum(8))
print(Sum(8,2))
```

______________________________________________

3. 下面代码的功能是,随机生成 50 个介于[1,20]的整数,然后统计每个整数出现频率。请把缺少的代码补全。

```
import random
x=[random.__①__(1,20) for i in range(__②__)]
r=dict()
for i in x:
    r[i]=r.get(i, __③__)+1
for k, v in r.items():
    print(k, v)
```

| 函　　数 | 描　　述 |
|---|---|
| seed(a=None) | 初始化随机数种子,默认值为当前系统时间 |
| random() | 生成一个[0.0, 1.0)之间的随机小数 |
| randint(a, b) | 生成一个[a,b]之间的整数 |
| getrandbits(k) | 生成一个 k 比特长度的随机整数 |
| randrange(start, stop[, step]) | 生成一个[start, stop)之间以 step 为步数的随机整数 |
| uniform(a, b) | 生成一个[a, b]之间的随机小数 |
| choice(seq) | 从序列类型(例如:列表)中随机返回一个元素 |
| shuffle(seq) | 将序列类型中元素随机排列,返回打乱后的序列 |
| sample(pop, k) | 从 pop 类型中随机选取 k 个元素,以列表类型返回 |

4. 下面代码是否能够正常执行,若不能,请解释原因;若能,请分析其执行结果。

```
from random import randint
result=set()
while True:
    result.add(randint(1,10))
if len(result)==20:
    break
print(result)
```

______________________________________________

5. 请根据下列代码画出相应图形。

```
import turtle
t=turtle.Pen()
for i in range(100):
```

```
    t.fd(i)
t.seth(i * 90)
```

______________________________________________

## 四、程序设计(每题 8 分,共 40 分)

1. 判断某一年是否为闰年的条件是:①如果年份能被 400 整除,则为闰年;②如果年份能被 4 整除但不能被 100 整除也为闰年。(5 分)

2. 判断三个正数能否构成三角形,若能构成三角形,则计算三角形的周长和面积。(5 分)

3. 编写程序,计算下列分段函数。(10 分)

$$y = \begin{cases} 0 & x < 0 \\ x^2 + 6 & 0 \leqslant x < 5 \\ 3x - 5 & 5 \leqslant x < 10 \\ 28 & x \geqslant 10 \end{cases}$$

4. 一个数如果恰好等于它的因子之和,这个数就称为"完数"。例如,6 的因子为 1、2、3,而 6=1+2+3,因此 6 是完数。编程,找出 1000 之内的所有完数,并输出这些完数。(提示:枚举法)(10 分)

5. 抛向空中一定高度的球,已知初始高度 H(米)和初始速度 V(米/秒),t 秒后球的高度为 $H+Vt-16t^2$(米)。请编程计算:(10 分)

(1) 球的最大高度。(提示:$t=V/32$ 秒时达到最大高度)

(2) 什么时候球会落地?(提示:每过 0.1 秒计算一次高度,当高度不再是一个正数时即可确定)

要求:

① 编写函数 getInput()从键盘输入初始高度 H 和初始速度 V。

② 利用函数 getInput()调用函数 isValid()保证输入的数值是正数。

③ 编写函数 Mheight()和 downTime()计算最大高度和落地时间。

④ 编写函数 Output()在屏幕上显示计算结果。

# 附录　各章参考答案及解析

## 第 1 章习题答案及解析

### 1. 简答题

(1) 列举几位对计算机发展产生过重要影响的人物,并简述他们的贡献。

**【答案】**

历史上有很多这样的人物,答案不唯一,如阿兰·图灵是一名数学家,发表了一篇经典论文《论可计算数及其在判定问题中的应用》。阿兰·图灵提出了"程序控制"思想,阐明了"有一种机器,也能像人脑一样执行指令序列",并且给出了一种计算模型,即著名的图灵机(Turing Machine)模型。

**【知识点】**

详见 1.1 节。

(2) 简述计算机采用二进制的原因。

**【答案】**

计算机是电子设备,利用计算机可以处理数字信号的数据,也可以处理模拟信号的数据。数字信号与模拟信号的区别如图 1-3(a)和图 1-3(b)所示。我们不难发现,数字信号的最大特点是容易实现,只要事先设定一个阈值,大于这个阈值,就认为这个信号表示"1",否则认为"0"。利用电子元器件较容易实现数字计算机,而实现模拟计算机需要较高精确度的电子元器件,实现是比较困难的,因此"模拟电子计算机"在计算机的发展史上稍纵即逝,很快被淘汰。我们目前使用的绝大多数计算机都是"数字电子计算机"。

**【知识点】**

详见 1.1 节。

(3) 什么是摩尔定律? 它描述的是什么现象?

**【答案】**

摩尔定律是由英特尔公司创始人之一戈登·摩尔(Gordon Moore)提出来的。其内容为:当价格不变时,集成电路上可容纳的元器件的数目,约每隔 18~24 个月便会增加一倍,性能也将提升一倍。

**【知识点】**

详见 1.1 节。

(4) 有 15 瓶药水,其中一瓶有毒。假如一只小白鼠喝下药水后是否中毒需要一个小时才会体现出来。如果有 4 只小白鼠,是否有办法用一个小时的时间检测出有毒的药水?

**【答案】**

4 位二进制数最多能表示 16 种状态,在本题中需要除去 0000。

【知识点】

详见 1.1 节。

【解析】

二进制数只有两个数字“0”或者“1”，按照“逢二进一”的原则计数，即每位计满 2 时向高位进 1。例如，二进制的数值“100”对应十进制的数值“4”，十进制的数值“10”对应二进制的数值“1010”。在本题中，解决的方案不唯一，有很多可行的办法。例如，在第 8、9、10、…、15 瓶药水中各取 1/4 给第一只小白鼠喝；在第 4、5、6、7、12、13、14、15 瓶药水中各取 1/4 瓶给第二只小白鼠喝；在第 2、3、6、7、10、11、13、14、15 瓶药水中各取 1/4 给第三只小白鼠喝；在第 2、3、5、7、9、11、13、15 瓶药水中各取 1/4 给第四只小白鼠喝。若第 2、3、4 只小白鼠死了，则表示第 7 瓶药水有毒，以此类推。

## 2. 填空题

(1) 基于冯·诺依曼思想而设计的计算机硬件由运算器、________、________、________和输出设备等 5 部分组成。

【答案】

存储器、控制器、输入设备

(2) 一个字节等于________位。

【答案】

8

【解析】

一个字节等于 8 比特，即一个字节等于 8 位。

(3) 世界上首台数字计算机诞生于________年。

【答案】

1946

(4) 程序设计语言的三个大类分别是________、________、________。

【答案】

机器语言，汇编语言，高级语言

【知识点】

详见 1.2 节。

## 3. 选择题

(1) 世界上第一台电子计算机是【　　】。

A) ENIAC　　B) ABC　　C) EDVAC　　D) Mark Ⅱ

【答案】

A

【知识点】

详见 1.1 节。

【解析】

ENIAC 诞生于 1946 年；ABC 研发于 1937—1941 年，但是不可编程；EDVAC 于 1949 年 8 月交付给弹道研究实验室，直到 1951 年 EDVAC 才开始运行；Mark Ⅱ 诞生于在 1947 年或 1948 年。

（2）最早计算机主要用于【　　】。

A）数据处理　　B）科学计算　　C）辅助设计　　D）过程控制

【答案】

B

【知识点】

详见 1.1 节。

（3）以下不同进制的四个数中，【　　】是最小的数。

A）$(101101)_2$　　B）$(52)_8$　　C）$(2B)_{16}$　　D）$(46)_{10}$

【答案】

B

【知识点】

详见 1.1 节。

【解析】

换算成十进制，A 为 45，B 为 42，C 为 43，D 为 46。

（4）设一个具有 20 位地址和 64 位字长的存储器，该存储器能存储【　　】的信息。

A）8MB　　B）4MB　　C）2MB　　D）1MB

【答案】

B

【知识点】

详见 1.1 节。

【解析】

$$(2^{20})\times 64/8/1024/1024=8(\text{MB})$$

（5）Python 是由【　　】发展来的。

A）C 语言　　B）ABC　　C）FORTRAN　　D）Pascal

【答案】

B

【知识点】

详见 1.3 节。

（6）英国科学家巴贝奇提出一种通用的计算机设计思想，称为【　　】。

A）加法器　　B）微机　　C）差分机　　D）分析机

【答案】

D

【知识点】

详见 1.1 节。

(7) 物理器件采用晶体管的计算机被称为【　　】计算机。

A) 第 1 代　　B) 第 2 代　　C) 第 3 代　　D) 第 4 代

【答案】

B

【知识点】

详见 1.1 节。

【解析】

第 1 代计算机基于真空管技术，第 2 代计算机采用晶体管制造，第 3 代计算机即第 3 代集成电路计算机，第 4 代计算机由大规模和超大规模集成电路组装成。

(8) 摩尔定律指出芯片上集成的晶体管数目每【　　】个月翻一番。

A) 6　　B) 12　　C) 18　　D) 27

【答案】

C

【知识点】

详见 1.1 节。

【解析】

当价格不变时，集成电路上可容纳的元器件的数目，约每隔 18～24 个月便会增加一倍。

(9) 下列不属于新型计算机的是【　　】。

A) 超导计算机　　B) 量子计算机

C) 半导体计算机　　D) DNA 计算机

【答案】

C

【知识点】

详见 1.1 节。

(10) 下列语言中，【　　】不是面向对象程序设计语言。

A) Java　　B) C 语言　　C) Python　　D) C#

【答案】

B

【知识点】

详见 1.2 节。

【解析】

C++ 在语言层面上提供了 OOP/GP 语法，但用 C 也可实现 OOP 思想；Java、C# 和 Python 都是支持 OOP 的。

(11) 程序中的错误主要分为语法错误和【　　】。

A) 逻辑错误　　B) 系统错误　　C) 自定义错误　　D) 结构错误

【答案】

A

【知识点】

详见 1.4 节。

(12) 程序可以不满足算法的【　　】性质。

A) 有外部量作为输入　　B) 产生至少一个输出

C) 指令清晰无歧义　　D) 指令执行次数有限

【答案】

D

【知识点】

详见 1.4 节

(13) 下列选项中,非通用编程语言的是【　　】。

A) C 语言　　B) SQL　　C) Java　　D) Python

【答案】

B

【知识点】

详见 1.2 节。

【解析】

SQL 是一种特定目的程序语言,但是它也含有过程式编程的元素;C、Java、Python 都是通用编程语言。

# 第 2 章习题答案及解析

## 1. 简答题

(1) 简述变量与常量的区别,并说明使用变量的必要性。

【答案】

在程序设计中,变量是指一个包含部分已知或未知数值或信息的储存位址,以及相对应的符号名称。通常使用变数名称参照储存值;将名称和内容(常量)分开能让被使用的名称独立于所表示的精确信息之外。

【知识点】

详见 1.2 节。

(2) 简述数据结构、数据类型和抽象数据类型的区别与联系。

【答案】

在程式设计的类型系统中,数据类型(data type)是用来约束数据的解释。在编程语言中,常见的数据类型包括原始类型(如整数、浮点数或字元)、抽象数据类型等;抽象数据类型(Abstract Data Type,ADT)是计算机科学中具有类似行为的特定类别的数据结构

的数学模型;或者具有类似语义的一种或多种程序设计语言的数据类型,如复数、容器。在计算机科学中,数据结构(data structure)是计算机中存储、组织数据的方式,如二叉树。

**【知识点】**

详见2.1节。

(3) 简述元组、列表、字典的不同和相同点。

**【答案】**

字典用“{ }”将需要处理的数据括起来,字典数据类型内存放的元素是“无序”的,这种数据类型定位元素值要依赖“键/值”,由键值元素值构成“元素对”,一个键/值对应一个元素值。

列表用“[ ]”将需要处理的数据括起来,列表数据类型内存放的元素是“有序”的,每个元素按照顺序排列,列表内的元素值可以修改。

元组用“( )”将需要处理的数据括起来,元组数据类型内存放的元素是“有序”的,与列表的主要区别是,元组初始化后元素值不能修改。

**【知识点】**

详见2.1.4节、2.1.5节、2.1.6节的元组、列表、字典的概念。

(4) 解释Python运算符/与//、* 与**的区别。

**【答案】**

“/”是除法运算;“//”是取整;“ * ”是乘法运算;“**”是乘方。

**【解析】**

详见2.3节。

## 2. 填空题

(1) 在Python中,常用的数据类型有数值类型、________、________、________、________、________和________等。

**【答案】**

字符串类型、布尔类型、列表类型、字典类型、元组类型、集合类型

**【知识点】**

详见2.1节。

(2) 使用math模块前,需要使用________语句导入该模块。

**【答案】**

import

**【知识点】**

详见2.2节。

(3) 数学表达式$\frac{\lambda^k}{k!}e^{-\lambda}$的Python表达式为________。



【答案】

形式不唯一，可以使用 math 库。例如：

```
pow(λ,k) * pow(math.e,- λ)/math.factorial(k)
```

【知识点】

详见 2.2 节。

(4) [2] in [1,2,3]返回的结果是________。

【答案】

False

【知识点】

详见 2.1 节。

【解析】

[1,2,3]是列表类型的数据。[2]也是列表类型的数据。两个列表类型的数据是不能比较的。如果要看某个数据是否在某个列表中，应该这样写：2 in [1,2,3]。看看 2 是否是包含在[1,2,3]这个列表中的数据。

(5) 命题“x 小于或等于 y，且大于 z”的 Python 表达式是________。

【答案】

x<=y and x>z

【知识点】

详见 2.1 节。

(6) 命题“x 小于或等于 y，或大于 z”的 Python 表达式是________。

【答案】

x<=y or x>z

【知识点】

详见 2.1 节。

(7) 命题“x 是 y 的倍数”的 Python 表达式是________。

【答案】

y%x==0

【知识点】

详见 2.2 节。

(8) 输入__name__返回的结果是________。

【答案】

通常是__main__

【知识点】

详见 2.4 节。

【解析】

“__name__”表示模块、类等的名字；“__main__”模块表示的是某个 py 类型文件。py 文件被直接执行时，对应的模块名就是__main__了。所以，当输入“__name__”时，反馈的是“__main__”结果。

(9) "BBJJTTUU"[::2]返回的结果是________。

**【答案】**

'BJTU'

**【知识点】**

详见 2.2 节。

**【解析】**

形如[ ]的索引中最多可以放两个冒号，意义是[start:stop:step]，用法非常灵活。"BBJJTTUU"[::2]是字符串"BBJJTTUU"中每隔 2 个字符取一个字符，即'BJTU'。

(10) 若字典 d={1:"a",2:"b"}，则 sum(d)返回的结果是________，sum(d.keys())返回的结果是________，sum(d.values())返回的结果是________。

**【答案】**

3，3，TypeError

**【知识点】**

详见 2.1 节。

### 3. 选择题

(1) 下列选项中合法的标识符是【　　】。

A) _　　B) class　　C) a&b　　D) 3x

**【答案】**

A

**【知识点】**

详见 2.4 节。

**【解析】**

B 是保留字；C 含有操作符；D 以数字开头。A 是合法的，字母、下画线、数字组合都可以构成标识符，但是头字母必须是下画线。

(2) 若 a=math.e ** (1j * math.pi)，则 1+a 输出的结果是【　　】。

A) 0　　B) 1　　C) 2　　D) 以上都不是

**【答案】**

D

**【知识点】**

详见 2.3 节。

**【解析】**

题中显然是欧拉公式，但 a 的值实际上在计算机中存储的并不是−1。

(3) 函数 type(pow(−1,0.5)−1j+0xf * 2.718)返回的结果是【　　】。

A) <class 'complex'>　　B) <class 'int'>

C) <class 'long'>　　D) <class 'float'>

**【答案】**

A

【知识点】

详见 2.1 节。

(4) len("BJTU")返回的结果是 4,len("北京交大")和 len("北京交大\nBJTU")返回的结果分别是【　　】。

A) 4,8　　B) 4,9　　C) 8,13　　D) 8,14

【答案】

B

【知识点】

详见 2.2 节。

【解析】

"\n"看上去是由反斜杠与字幕 n 组成的,但"\n"是转义字符,实际上只有一个长度。

(5) 若字符串 s="BeijingJiaoTongUniversity",与 s[0:-1]不仅输出结果相同而且具有相同含义的是【　　】。

A) s[:-1]　　B) s[:]

C) s[:len(s)-1]　　D) s[:len(s)]

【答案】

A

【知识点】

详见 2.2 节。

【解析】

A 与 C 输出相同,区别在于 A 是倒序,C 是顺序。

(6) 设元组 t=(2,3),则 a[1]=1 返回的结果是【　　】。

A) (2,1)　　B) (1,3)　　C) TypeError　　D) NameError

【答案】

A

【知识点】

详见 2.2 节。

【解析】

元组类型的元素是不可改变的。

(7) 设列表 l=[3],则 l*3 返回的结果是【　　】。

A) [9]　　B) TypeError　　C) [3],[3],[3]　　D) [3,3,3]

【答案】

D

【知识点】

详见 2.2 节。

【解析】

l*3 的含义是对列表中的元素重复 3 次,即[3,3,3]。

(8) 以下会返回错误的语句是【　　】。

A) d1={}　　B) d2={0:1}

C) d3=dict([0,1],[2,3])　　D) d4=dict(([0,1],[2,3]))

【答案】

C

【知识点】

详见 2.1 节中的字典类型。

【解析】

"{}"表示一个空字典;"{0:1}"表示键和对应的键值;"dict(([0,1],[2,3]))"表示{0: 1,2: 3}。"dict([0,1],[2,3])"键值不能是列表类型。

(9) 表达式 25//3-100//5**2%3*5 的值为【　】。

A) 0　　B) 8　　C) 3　　D) 1

【答案】

C

【知识点】

详见 2.3 节。

(10) 下列表达式中与数学表达式 $\frac{ab}{cd}$ 不对应的是【　】。

A) a*b/c/d　　B) a*b/(c*d)　　C) a/c*b/d　　D) a*b/c*d

【答案】

B

【知识点】

详见 2.3 节。

【解析】

写表达式的时候应该多使用括号,尤其是复杂的表达式。

(11) 下列表达式非法的是【　】。

A) 71//7　　B) 71.7//1.7　　C) 1+7j/3j　　D) 3j/j

【答案】

D

【知识点】

详见 2.1 节。

【解析】

单独出现的 j 会被认为是变量,一个虚数单位应该使用 1j 表示。"1+7j/3j"的结果是(3.3333333333333335+0j)。

(12) 表达式 type(range(9))返回的结果是【　】。

A) <class 'range'>　　B) <class 'list'>

C) <class 'tuple'>　　D) <class 'str'>

【答案】

A

【知识点】

详见 2.1 节。

(13) 若 list1＝["name","address","postcode"],list2＝["BJTU","Haidian","100044"],下列能使上述列表转换为以 list1 中元素为键,以 list2 中元素为值的字典的语句是【　　】。

A) dict(list1,list2)　　　　B)dict((list1,list2))

C) dict(zip(list1,list2))　　　　D) dict(key＝list1,value＝list2)

**【答案】**

C

**【知识点】**

详见 2.1 节。

**【解析】**

dict 并没有 key 和 value 参数。而 zip()是 Python 的一个内建函数,它接受一系列可迭代的对象作为参数,将对象中对应的元素打包成一个元组,然后返回由这些 tuples 组成的列表。

(14) 若字典 d＝{1:"a",2:"b"},则 len(d)返回的结果是【　　】。

A) 10　　B) 2　　C) 6　　D) 4

**【答案】**

B

**【知识点】**

详见 2.2 节。

**【解析】**

字典中有两个键/值对,因此长度为 2。

(15) 若字典 d＝{1:"a",2:"b"},则能够访问 d 的第一个元素的语句是【　　】。

A) d[0]　　B) d[1]　　C) d["0"]　　D) d["1"]

**【答案】**

B

**【知识点】**

详见 2.2 节。

**【解析】**

字典的元素是键/值对,字典本身是不可索引的,但可以用 list 函数转化为列表类型;严格说,字典本身其实是无序的,因此第一个元素这种说法也是有问题的。

(16) 若集合 s＝{1,1,1,2,2,2,3,3,3},则 sum(s)和 len(s)的返回值是【　　】。

A)18,9　　B) 6,3　　C) 18,3　　D) 6,9

**【答案】**

B

**【知识点】**

详见 2.1 节。

【解析】

实际上 s 是{1,2,3},集合(set)是一个无序不重复元素的序列。基本功能是进行成员关系测试和删除重复元素。

## 第 3 章习题答案及解析

### 1. 选择题

(1) 下列操作能够创建文件对象的是【　　】。

A) open　　B) file　　C) create　　D) make

【答案】

A

【知识点】

详见 3.2 节。

【解析】

Python 对文件进行操作时先要打开文件,以获得文件的控制权。打开文件时会创建文件对象,因此用 open()函数打开文件时可以创建文件对象,答案为 A。

(2) 下列操作不能够读取文件的是【　　】。

A) read　　B) readline　　C) readlines　　D) readall

【答案】

D

【知识点】

详见 3.2 节。

【解析】

Python 读取文件有 3 种方法,其中 read()可以读取整个文件内容,readline()可以逐行读取文件内容,readlines()可以将读取的文件内容保存为列表。但没有 readall()操作,因此答案为 D。

(3) 关于语句 f=open("a.txt","w+"),下列说法正确的是【　　】。

A) 只能读取数据　　B) 只能写入数据

C) 文件必须已经存在　　D) 文件可以不存在

【答案】

D

【知识点】

详见 3.2 节。

【解析】

“w+”表示“同时读写文件”模式,如果文件不存在则创建 a.txt 文件,存在则覆盖原文件内容。因此答案为 D。

(4) 下列程序的输出结果是【　　】。

```
f=open("w.txt","w+ ")
f.write("Lux et Veritas")
f.seek(7)
s=f.read(3)
f.close()
print(s)
```

A) eri　　B) Ver　　C) tas　　D) Lux

【答案】

B

【知识点】

详见3.2节。

【解析】

程序创建了一个w.txt文件,内容为“Lux et Veritas”。seek(7)的功能是将文件指针从起始位置移动7位,指向“V”(这里请注意“空格”也占一位)。此时读取文件时会从文件指针指向的位置,即“V”开始读取文件。read(3)的功能是读取3个字节,因此s=“Ver”,答案为B。

(5) 下列程序的输出结果是【　　】。

```
f=open("w.txt","w")
f.write("Lux et Veritas")
f.close()
f=open("w.txt","r")
f.read(3)
f.seek(4,1)
print(f.tell())
```

A) 5　　B) 6　　C) 7　　D) 8

【答案】

C

【知识点】

详见3.2节。

【解析】

程序创建了一个w.txt文件,内容为“Lux et Veritas”。read(3)的功能是读取文件前3个字节的内容,即“Lux”,此时文件指针指向“x”后面的空格。seek(4,1)的功能是将文件指针从当前始位置移动4位,此时会指向“V”(这里请注意“空格”也占一位)。这时f.seek(4,1)会返回文件指针相对于文件起始位置的偏移量。而“V”相对于文件起始位置的偏移量是7,因此答案为7,选C。

(6) 执行下列语句会报错,错误在第【　　】行。

```
f=open("w.txt","w+")
f.write("Lux et Veritas")
f.seek(4,1)
```

A) 1　　　　　B) 2　　　　　C) 3　　　　　D) 并不会报错

**【答案】**

C

**【知识点】**

详见 3.2 节。

**【解析】**

程序创建了一个 w.txt 文件,写入内容为“Lux et Veritas”。此时文件指针在文件结尾处。而 seek(4,1)将文件指针从当前始位置移动 4 位,超出文件范围,会报错,因此答案为 C。

## 2. 程序设计题

(1) 编写程序,让用户输入自己的姓名、年龄、最喜欢的颜色和最喜欢的书。程序将这四项内容保存在一个文本文件中,每一项内容分别放在单独一行上,再逐行显示文件内容。

**【答案】**

```
file='my_file.txt'
ls=[]
name=input('请输入你的姓名:')
ls.append('我的姓名:'+name)
age=input('请输入你的年龄:')
ls.append('\n我的年龄:'+age)
color=input('你最喜欢的颜色是:')
ls.append('\n我最喜欢的颜色:'+color)
book=input('你最喜欢的书是:')
ls.append('\n我最喜欢的书:'+book)
My_file=open(file,'w+')          #以同时读写模式打开文件
My_file.writelines(ls)           #添加新行
My_file.seek(0)
for line in My_file:
    print(line)
My_file.close()
```

**【知识点】**

详见 3.2 节。

**【解析】**

该题主要锻炼学生“写文件”的方法。利用 writelines()方法将列表 ls 写入文件。列表中的每个元素就是文件中表示每行内容的字符串,为实现逐行保存,在每行字符串前添

加“\n”换行符。“w＋”模式可以在写入文件的同时读取文件内容。写文件结束时文件指针在文件结尾处，为显示文件内容，需要将用 seek(0)将文件指针移动到文件起始位置。

（2）编写程序自动造句，每个句子包含 4 个部分：形容词＋名词＋动词＋名词。例如，聪明的阿凡提骑毛驴（提示：这些词汇可以事先保存在 4 个不同的文件中，程序随机选择一个词组成句子）。

**【答案】**

```
import random
xrc=['聪明的','狡猾的','勤劳的','善良的','弱小的','强大的']
mc=['猎狗','狗熊','贝利','警察','大象','猴子']
dc=['吃','追','踢','抓','偷','搬']
mc2=['香蕉','狐狸','蜂蜜','木头','坏人','足球']
My_file=open('xrc.txt','w')
for line in xrc:
    My_file.write(line+'\n')
My_file.close()
My_file=open('mc.txt','w')
for line in mc:
    My_file.write(line+'\n')
My_file.close()
My_file=open('dc.txt','w')
for line in dc:
    My_file.write(line+'\n')
My_file.close()
My_file=open('mc2.txt','w')
for line in mc2:
    My_file.write(line+'\n')
My_file.close()
My_file=open('xrc.txt','r')
xrcls=My_file.readlines()
My_file.close()
My_file=open('mc.txt','r')
mcls=My_file.readlines()
My_file.close()
My_file=open('dc.txt','r')
dcls=My_file.readlines()
My_file.close()
My_file=open('mc2.txt','r')
mc2ls=My_file.readlines()
My_file.close()
sentence_xrc=''.join(random.sample(xrcls,1)).strip('\n')
sentence_mc=''.join(random.sample(mcls,1)).strip('\n')
sentence_dc=''.join(random.sample(dcls,1)).strip('\n')
sentence_mc2=''.join(random.sample(mc2ls,1)).strip('\n')
sentence=sentence_xrc+sentence_mc+sentence_dc+sentence_mc2
print(sentence)
```

【知识点】

详见3.2节。

【解析】

首先定义4个词语列表,再采用write()方法创建4个词语文件,将词语列表中的词语逐行写入文件中。造句时,首先用readlines()方法读取文件,将词语存入一个列表中,再从列表中随机选取一个词语,合并组成一个完整的句子。为能够随机选取词语列表中的词语,使用了random库的sample()方法。由于readlines()方法生成的词语列表中每个元素是一行内容,后面带有换行符"\n",因此要用strip('\n')方法消除换行符。但strip()方法只适用于字符串,而词语列表中的元素是列表类型,因此用''.join方法将列表元素转换为字符串,再用strip('\n')方法消除换行符。

(3) Python学习笔记:在文本编辑器中创建一个新文件,写几句话总结你至今学到的Python知识,其中每句话占一行,内容是"我可以用Python……"。保存这个文件。编写程序,读取该文件并打印三次:第一次打印时读取整个文件;第二次打印时遍历文件对象;第三次逐行读取内容,并用方法replace()将"Python"替换成另外一门编程语言的名称,如C、Java等,将修改后的各行都打印到屏幕上。

【答案】

```
file='python.txt'
My_file=open(file, 'r')
for line in My_file.readlines():
    print(line)
My_file.seek(0)
for line in My_file:
    print(line)
My_file.seek(0)
for line in My_file:
print(line.replace('Python', 'C'))
My_file.seek(0)
for line in My_file:
print(line.replace('Python', 'JAVA'))
My_file.close()
```

【知识点】

详见3.2节。

【解析】

该题主要让学生掌握"读文件"的方法。要注意的是每次读取文件后文件指针会移动到文件结尾。再次读文件时,需要用seek(0)将文件指针移动到文件起始位置。

(4) 访客名单:编写一个while循环,提示用户输入姓名。用户输入姓名后,在屏幕上打印一句问候语,并将一条问候记录添加到文件guest_book.txt中。要求每条记录独占一行。

【答案】

```
while True:
    name= input('请输入你的姓名：')
    book= name+ ',你好！ 欢迎光临！ '
    print(book)
    My_file= open('guest_book.txt','a')
    My_file.write(book+ '\n')
My_file.close()
```

【知识点】

详见 3.2 节。

【解析】

该题利用 while 循环结构可以重复提示用户输入姓名。注意在写入文件时要用"a"追加写模式，避免新写入的访客记录覆盖原有记录。

## 第 4 章习题答案及解析

### 1. 简答题

(1) 什么是算法？请从日常生活、学习和工作中找出两个例子，描述它们对应的算法。

【答案】 略

【知识点】

详见 4.1 节。

【解析】

算法(Algorithm)是指解题方案的准确而完整的描述，是一系列解决问题的清晰指令，算法代表着用系统的方法描述解决问题的策略机制。答案中两个例子所对应的算法清晰地体现了算法的五类特征：有穷性、确定性、有零个或多个输入、有一个或多个输出、有效性。

(2) 阐述程序设计的三种基本结构的特点；在此基础上，设计两种基本结构(基于基本结构的特点)。

【答案】 略

【知识点】

详见 4.1 节。

【解析】

顺序语句由语句序列组成，程序执行时，按 照语句的顺序，从上而下，一条一条地顺序执行；选择结构表示程序的处理步骤出现了分支，它需要根据某一特定的条件选择其中的一个分支执行；环结构表示程序反复执行某个或某些操作，直到某条件为假(或为真)时才可终止循环；此外，所设计的两种基本结构满足四个条件：只有一个入口、只有一个出口、结构内的每一个部分都可能被执行到、结构内不存在死循环，详见 4.1 节。

（3）解释算术运算、关系运算和逻辑运算。

【答案】 略

【知识点】

详见 4.2 节。

【解析】

算术运算符、关系运算符和逻辑运算符是分别用来处理四则运算的符号、比较运算的符号和逻辑运算的符号，对应连接的表达式称为算术运算、关系运算和逻辑运算。

（4）Python 语言中如何表示“真”和“假”？

【答案】

True 和 False

【知识点】

详见 4.2 节。

【解析】

逻辑表达式的值应该是一个逻辑“真”或“假”。Python 语言编译系统在给出逻辑运算结果时，以“True”代表“真”、以“False”代表“假”，但也需要注意到在 Python 语言中，非“0”的数值或者字符串被视为真“True”，例如“2 and 3”的值等于“3”，详见 4.2 节。

（5）传统流程图和 N-S 流程图各有何特点？其各自优势体现在哪里？

【答案】

传统的流程图表示的程序结果更加清晰；而 N-S 流程图表示的程序更加简洁明了。

【知识点】

详见 4.1 节。

【解析】

传统的流程图用流程线表示各框的执行顺序，对流程线的使用没有严格限制。相对于传统流程图，N-S 流程图废除了流程线，整个算法结构是由各个基本结构按顺序组成，N-S 流程图中的上下顺序就是执行时的顺序，也就是图中位置在上面的先执行、位置在下面的后执行，更适合结构化程序设计，详见 4.1 节。

## 2. 计算题

（1）设 x=4，y=5，z=6，计算 x+y>z and x==z。

（2）设 x=4，y=5，z=6，计算 x or y or z。

（3）设 x=4，y=5，z=6，计算 x and y and z。

（4）设 x=4，y=5，z=6，计算 not x>y and not (x>z)。

（5）设 x=4，y=5，z=6，计算(y>x) and (z>x) and 0。

【答案】

（1）False

（2）4

（3）6

（4）True

(5) 0

**【知识点】**

详见 4.2 节。

**【解析】**

逻辑表达式的值应该是一个逻辑“真”或“假”。Python 语言编译系统在给出逻辑运算结果时，以“True”代表“真”、以“False”代表“假”，但也需要注意到在 Python 语言中，非“0”的数值或者字符串被视为真“True”，例如“2 and 3”的值等于“3”，详见 4.2 节。

### 3. 程序设计题

(1) 有 3 个杯子(记为 A、B 和 C)，其中 A 盛放的是苹果汁、B 盛放的是西瓜汁、C 是空杯，要求 A 和 B 互换。

**【知识点】**

详见 4.1 节。

**【解析】**

该问题的核心就是实现两个变量值的交换，其中“C”表示一个中间变量，首先将“A”的值赋给中间变量“C”，然后将“B”的值赋给“A”，最后将“C”赋给“B”，这样就实现了“A”和“B”交换的目的。

(2) 依次输入 5 个数，将其升序排列。

**【知识点】**

详见 4.1～4.4 节。

**【解析】**

针对这一排序问题，首先需要选择排序算法，典型的算法有冒泡排序、选择排序和快速排序等(可通过互联网查阅相关算法)，然后基于排序算法确定具体每一次比较的分支语句，最后通过相应的表达式加以实现。

(3) 判断某一个数是否是完数(完数就是其所有真因子的和，恰好等于它本身)。

**【知识点】**

详见 4.2 节和 4.3 节。

**【解析】**

首先需确定某一整数的所有真因子(即除了自身以外的约数)，例如 6 的真因子为 1、2 和 3；然后通过 if 语句、算术表达式和逻辑表达式判断是否是完数，例如 1、2 和 3 的和是 6，就表示 6 为一完数。

(4) 求两个正整数的最大公因子。

**【知识点】**

详见 4.1～4.4 节。

**【解析】**

针对两个正整数的最大公因子这一问题展开分析，首先确定其求解算法为辗转相除法，然后通过递归加以实现或者通过 while 循环控制语句加以实现。

(5) 统计某一字符串中出现 2 次的字母。

【知识点】

详见 4.2～4.4 节。

【解析】

确定某字符串出现 2 次的字母，通过 for 循环控制语句含义，遍历这一字符串；接下来基于 if 语句来判断出现的字母是否属于 26 个字母，最后统计出现 2 次的所有字母。

（6）2017 年某银行的定期存款利率表如下：

| 类型年 | 利率/% | 类型年 | 利率/% |
|---|---|---|---|
| 三个月 | 1.1 | 两年 | 2.1 |
| 半年 | 1.3 | 三年 | 2.3 |
| 一年 | 1.5 | 五年及以上 | 2.8 |

现在存入 100000 元，五年后本金加利息各是多少？

【知识点】

详见 4.3 节和 4.4 节。

【解析】

基于多分支结构的 if-elif-else 语句，将问题分为 5 种类型，进一步通过循环控制语句计算五年后本金加利息。

（7）给定一个 n（n＞2）阶实数方阵，确定矩阵元素中的最大值。

【知识点】

详见 4.1～4.4 节。

【解析】

在 Python 语言程序设计中，首先通过列表来表示矩阵，也就是列表中的元素也是列表，比如[[1,2,3],[4,5,6],[7,8,9]]则可以表示矩阵

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

接下来通过 for 语句遍历列表，则可基于关系表达式和 if 语句获得矩阵元素的最大值。

（8）在 1849 年，阿尔方·德·波利尼亚克提出了一般的猜想：对所有自然数 k，存在无穷多个素数对（p，p＋2k）。k＝1 的情况就是孪生素数猜想。孪生素数猜想是一大数学难题，在近一个世纪以来，一大批数学家在努力解决这一伟大的猜想。请编程计算 100 到 10000 内有多少孪生素数对？

【知识点】

详见 4.1～4.4 节。

【解析】

首先通过遍历循环确定 100～10000 中的所有素数，然后利用关系表达式、循环控制语句和分支语句确定满足条件的素数对。

(9) 假如一个班级有 30 名学生，这当中两个人同一天出生的概率是多大？（提示：用 randint 函数来生成随机的生日，这个函数包含在 random 模块中）

**【知识点】**

详见 4.1～4.4 节。

**【解析】**

首先调用 random 函数模块，确定 30 名学生的生日，然后基于循环语句、分支语句和关系表达式来确定两个人同一天出生的样本数，最后输出两个人同一天出生的概率。

(10) 用牛顿法求解下面方程在 1.5 附近的根。

$$2x^3 - 4x^2 + 3x - 6 = 0$$

**【知识点】**

详见 4.1～4.4 节。

**【解析】**

首先通过互联网或者数值分析等书籍，查阅牛顿法的算法原理和迭代步骤，然后利用循环语句和分支语句给出其解的迭代步骤，最后输出符合条件的解。

(11) 通过至少 3 种不同的排序算法实现对 6,3,7,8,5 降序排列(排序算法可自行构造或者通过互联网搜索)。

**【知识点】**

详见 4.1～4.4 节。

**【解析】**

实现排序的核心是算法的设计，可基于枚举的思想设计冒泡排序算法、基于分治的思想设计快速排序算法，进一步可通过分支语句和循环控制语句加以实现。

(12) 当输入为 5 时，写出下面程序运行结果。

```
import math
def fun(num):
    for j in range(2,math.floor(num/2)+1):
        if num%j==0:
            return False
        else:
            return True
def main():
    n=eval(input('Please input an integer:'))
    c=0
    for i in range(2,n+1):
        if fun(i):
            c+=1
        print(c)
main()
```

**【知识点】**

详见 4.2～4.4 节。

**【解析】**

阅读以上程序,可看出该程序包含了三个函数文件,一是 math 库的函数 floor(该函数的调用给出一种函数库的使用方法);二是定义了函数 fun();三是定义了函数 main(),同时该函数调用了 fun(),通过函数的嵌套调用,可得到问题的解。

(13) 写出下面程序的运行结果。

```
x='BJTU'
for i in x:
    for j in range(1,3):
        if i=='J':
            continue
        else:
            print(i,end='')
```

**【知识点】**

详见 4.2～4.4 节。

**【解析】**

该程序主要是对 for 循环语句和 continue 的使用,程序的运行结果 BBTTUU 说明 continue 语句用来结束当前当次循环。

(14) 打印输出一个 10×10 的乘法表。

**【知识点】**

详见 4.4 节。

**【解析】**

一个 10×10 的乘法可通过两层循环控制语句加以实现,同时为了保持乘法表的格式整齐规范,在输出中需合理地使用 format 格式、end 和\n 等输出命令。

(15) 编写程序输出以下图形。

```
         *
        ***
       *****
      *******
     *********
    ***********
   *************
  ***************
 *****************
*******************
```

**【知识点】**

详见 4.2 和 4.4 节。

**【解析】**

该问题主要是依赖于两层循环控制语句的使用,同时为了满足输出的图形要求,要合

理地使用 end 等基本命令。

# 第 5 章习题答案及解析

## 1. 简答题

(1) 简述使用函数的原因和意义。

**【答案】**

函数可以实现代码复用,提高编程效率。使用函数可以简化程序结构。复杂的编程问题可以分解成若干简单的子问题,每个子问题用函数来解决。函数像积木块一样通过灵活组合构建复杂程序。这种简化问题的方法体现了“自顶向下、模块化编程”的程序设计思想。使用函数可以使程序更容易阅读。函数使程序结构层次分明,好的函数名可以直接体现程序的功能,易于阅读和理解。使用函数还可以让程序更容易调试和测试。

**【知识点】**

详见 5.1 节。

**【解析】**

函数是实现某特定功能的程序块,这使得函数可以成为复杂程序的组成模块,并带来容易修改、便于调试等一系列优点。因此 Python 程序设计离不开函数。

(2) 什么是匿名函数? 请举例说明。

**【答案】**

匿名函数使用 lambda 关键字定义,省略了 def 声明函数的标准步骤,因此得名。例如:lambda x,y: x+y。

**【知识点】**

详见 5.1 节。

**【解析】**

匿名函数的本质是一个表达式,它能接收任何数量的参数并返回表达式的值。如上例中 x、y 是参数,表达式是 x+y,返回的是二者的和。从表达式的角度理解匿名函数更容易理解和掌握。

(3) 什么是递归函数? 请举例说明。

**【答案】**

函数可以调用其他函数,也可以调用自身,这样的函数称为递归函数。例如阶乘函数是典型的递归函数。其定义如下:

```
def fact(n):
    if n==0:
      return 1
    else:
      return n* fact(n-1)
```

【知识点】

详见 5.5 节。

【解析】

递归通常用来解决结构相似的问题，即将复杂问题不断分解成形式和结构与自身类似的子问题，直到子问题可以直接求解。递归函数逻辑清晰，结构简单，可以提高程序的可读性。

（4）使用哪个关键字创建函数？

【答案】

Python 用关键字 def(def 是 define 的缩写)创建函数。

【知识点】

详见 5.1 节。

【解析】

函数定义有规范格式，其中 def 是关键字。创建函数时 Python 会为该函数分配内存空间，保存其中的变量和结果。

（5）如何调用函数？

【答案】

输入函数的名字并传递相应的参数值就可以调用函数。

【知识点】

详见 5.1 节。

【解析】

Python 调用函数时直接输入函数名称并给出具体的参数值就可以。

（6）如何向函数传递参数？

【答案】

函数有 3 种方法将实参传递给形参：按照位置传递参数、按照关键字传递参数、按照默认值传递参数。在调用函数时，首先按位置顺序传递参数，其次按关键字传递参数。多余的非关键字参数传递给元组，多余的关键字参数传递给字典。

【知识点】

详见 5.2 节。

【解析】

函数的参数传递方法很多，默认方式是按照位置传递参数。各种参数传递方式可以混合使用，这时要遵循相应的规则，才能正确传递参数。

（7）函数最多可以有多少个参数？

【答案】

函数的参数数量没有限制。

【知识点】

详见 5.2 节。

【解析】

Python 支持可变数量的参数传递，因此函数的参数数量是没有限制的。但参数过多

可能会使函数的定义和调用复杂化,从程序的可读性以及执行效率的角度看未必合适。因此在定义函数时参数设置应该以“少而精”为原则。

(8) 如何从函数返回结果?

**【答案】**

用 return 语句返回结果。return 语句是可选项,可以在函数体内任何地方出现,表示函数执行到此结束,控制权返回给调用程序,同时返回处理结果。如果没有 return 语句,函数会自动返回 None。

**【知识点】**

详见 5.3 节。

**【解析】**

return 语句是可选项,主要功能是返回函数运行结果。返回的数据类型除常见的数值、字符串外,还可以是布尔值或列表等。当函数中有多条 return 语句时,执行完第一条 return 语句就会退出函数,不再执行其他 return 语句。函数中如果没有 return 语句,会自动返回 None,表示没有返回值。它的数据类型是 NoneType。

(9) 下面的函数有返回值吗?

```
def Hello():
    print("Hello Python")
```

**【答案】**

该函数没有 return 语句,函数会自动返回 None。

**【知识点】**

详见 5.3 节。

**【解析】**

当函数中没有 return 语句时,会自动返回 None,表示没有返回值。它的数据类型是 NoneType。函数的执行结果是在屏幕上显示“Hello Python”,但这不是函数的返回值。

(10) 函数运行结束后,函数中的局部变量会发生什么变化?

**【答案】**

函数中的局部变量会消失,不能再被访问和使用。

**【知识点】**

详见 5.4 节。

**【解析】**

Python 内部有内存管理(memory management)机制,在创建一个函数时会为其分配内存空间,当函数运行完毕后会释放这个内存空间,相应地,函数内部创建的各种变量,包括保存运行结果的变量会随之消失。因此函数中的局部变量只能在函数内部使用和访问。

(11) 如果希望在函数中修改全局变量的值,需要使用什么关键字?

**【答案】**

用关键字 global 修改。

【知识点】

详见 5.4 节。

【解析】

通常情况下,全局变量不能在函数内部进行修改,局部变量不能在函数外访问。如果想在函数内部修改变量,要用关键字 global 将这个变量声明为全局变量,这样才能修改它,并且在函数外可以访问修改过的变量。

(12) 递归必须满足什么条件?

【答案】

递归有两个基本条件:

① 基例:确定递归何时终止,即何时终止分解子问题,也称为递归出口。

② 递归模式:即复杂问题如何分解为子问题,也称为递归体。

【知识点】

详见 5.5 节。

【解析】

递归是一种重要的编程方法,通常用来解决结构相似的问题,其基本思路是将一个复杂问题转化成一个或几个子问题,子问题的形式和结构与原问题相似,但规模小,再把这些子问题进一步分解成规模更小的子问题,直到每个子问题可以直接求解。基例就是分解的最小子问题,它已经可以直接求解,不必继续分解。递归模式就是复杂问题与子问题共同具有的相似结构,例如阶乘和幂运算等。只有具备了这两个要素,才能保证递归函数在有限次计算后得出结果。

(13) 下列程序的输出结果是什么?

```
x=7
def main():
    x=5
    f()
    print(x)
def f():
    print(x)
main()
```

```
x=7
def main():
    global x
    x=5
    f()
    print(x)
main()
```

【答案】

左边的程序运行结果是:7、5

右边的程序运行结果是:5、5

【知识点】

详见 5.4 节。

【解析】

左边程序中的函数 f()功能是显示变量 x,由于变量 x 在函数 f()中没有定义,因此会使用函数外的全局变量 x=7,因此 main()中函数 f()的结果是 7。main()内的 print(x)

语句中的变量 x 是局部变量，main()定义了这个变量 x=5，因此 print(x)的结果是 5，程序最后的执行结果是 7、5。右边的程序与左边程序的区别在于 main()将变量 x 定义为全局变量，并且修改其值为 5，这样 f()和 print(x)中的变量 x 都为 5，因此程序执行结果是 5、5。

## 2．填空题

(1) 在函数内部可以通过关键字________来定义全局变量。

**【答案】**

global

**【知识点】**

详见 5.4 节。

(2) 一个没有 return 语句的函数的返回结果是________。

**【答案】**

None

**【知识点】**

详见 5.3 节。

(3) return [1,2],{3,4},5 语句返回的类型是________。

**【答案】**

元组

**【知识点】**

详见 5.3 节。

(4) 请使用 lambda 表达式表示下列函数________。

```
def fun(x,y=4)
    return y* x
```

**【答案】**

lambda x,y=4：y * x

**【知识点】**

详见 5.1 节。

## 3．选择题

(1) 函数的实际参数可以是【　　】。

A) 变量　　B) 常量　　C) 函数　　D) 以上都行

**【答案】**

D

**【知识点】**

详见 5.1 节。

**【解析】**

Python 函数的实参可以是常量、变量、表达式、函数等，无论何种类型，在进行函数调

用时都必须具有确定的值，才能让函数运行。

(2) 函数 isinstance(1+0j,int)返回的结果是【　　】。

A) True　　B) False　　C) SyntaxError　　D) TypeError

**【答案】**

B

**【知识点】**

详见 5.1 节。

**【解析】**

函数 isinstance(object,type)是 Python 的内建函数，功能是判断给定的对象是否是给定的类型。如果是，则返回 True，否则返回 False。1+0j 是复数，不是整数类型，因此返回 False，答案为 B。

(3) 下列函数中，可以用于将数字转换成字符的是【　　】。

A) ord()　　B)chr()　　C) oct()　　D) hex()

**【答案】**

B

**【知识点】**

详见 5.1 节。

**【解析】**

这 4 个函数中，ord()取 Unicode 码，oct()和 hex()是将输入数字转换为对应的八进制和十六进制数，chr()是将数字转换为字符，因此答案为 B。

(4) 下列语句中，正确的是【　　】。

A) def f(a=0,b):　　B) def f(a,b==0):

C) def f(a,b,*):　　D) def f(a,*b):

**【答案】**

D

**【知识点】**

详见 5.2 节。

**【解析】**

A 中默认参数 a=0 应该放到位置参数 b 的后面。B 中默认参数的定义方式不对，不应该用“==”，而应该用“=”。C 中参数定义形式不对，“*”应该放到参数 a 或 b 前面，表示该参数是数量可变的参数。D 是正确的，答案为 D。

(5) 执行下列程序段返回的结果是【　　】。

```
def f(*a):
    print(type(a))
if f(9,9): True
else: None
```

A) True　　B) False

C) None　　D) <class 'tuptle'>

**【答案】**

D

**【知识点】**

详见 5.2 节。

**【解析】**

可变数量的参数,实质是元组参数,返回 tuptle 类型。

(6) 下列定义的匿名函数能够返回两个数中较大的是【　　】。

A) fmax=lambda x,y:x if x>y else y

B) fmax=lambda x,y:x if: x>y else: y

C) fmax=lambda x,y: if x>y,x else y

D) fmax=lambda x,y: if: x>y,x else: y

**【答案】**

A

**【知识点】**

详见 5.1 节。

**【解析】**

此题考察 Lambda 表达式的格式,A 是正确的。

(7) 下列程序段返回的结果是【　　】。

```
a="first"
def second(a):
    a="second"
def third():
    global a
a="third"
third()
print(a,end=',')
second("fourth")
print(a)
```

A) first,second　　B) second,third

C) third,third　　D) second,first

**【答案】**

C

**【知识点】**

详见 5.2 节。

(8) 下列关于函数的说法正确的是【　　】。

A) 函数定义时必须有形参　　B) 函数定义时必须有 return 语句

C) 函数的形参和实参应该同名　　D) 以上都是错的

【答案】

D

【知识点】

详见 5.1 节。

【解析】

有些函数在运行时不需要参数，因此定义时不会定义形参。return 语句是可选项。形参通常表示函数参数功能或物理含义，而实参通常是实际数值、变量、表达式或函数，二者不需要同名。因此答案是 D，前述 3 种说法都不对。

(9) 若匿名函数 f=lambda x,y: x+y，则 f({1,2},{3,4})返回的结果是【　　】。

A) {1,2,3,4}　　B) {4,6}　　C) TypeError　　D) SyntaxError

【答案】

C

【知识点】

详见 5.1 节。

【解析】

该函数求 x+y，此时 x={1,2}，y={3,4}是集合，集合没有“+”这种操作，因此会提示 TypeError，答案是 C。

(10) 若匿名函数 f=[lambda x=3: x * 3,lambda x: x**3]，则 f[1](f[0]())返回的结果是【　　】。

A) 728　　B) 729　　C) TypeError　　D) SyntaxError

【答案】

B

【知识点】

详见 5.1 节。

【解析】

f 是一个列表，f[1]=lambda x: x**3，这是一个匿名函数，参数 x=f[0]()。f[0]=lambda x=3:x * 3，也是一个匿名函数，如果没有输入参数，则 x 的值是默认值 3，因此 f[0]()=3 * 3=9。f[1](9)=9**3=729，答案为 B。

(11) 若 def f(): pass，则 type(f())返回的结果是【　　】。

A) <class 'NoneType'>　　B) <class 'bool'>

C) <class 'function'>　　D) 以上都不是

【答案】

A

【知识点】

详见 5.1 节。

(12)下列语句中返回值为 5.0 的是【　　】。

A) math.ceil(4.2)　　B) math.floor(5.3)

C) round(5.3)　　D) 以上都不是

【答案】

D

【知识点】

详见 5.1 节。

【解析】

前 3 项返回的都是整数 5，不是浮点数 5.0。

（13）若匿名函数 f=[lambda x=2：x * 2，lambda x=2：x**2]，则 f[2](f[1]())返回的结果是【　　】。

A）16　　B）24　　C）IndexError　　D）SyntaxError

【答案】

C

【知识点】

详见 5.1 节。

【解析】

f 是一个列表，包含 2 个元素，下标从 0 开始，因此不存在 f[2]这个元素，所以程序会提示“IndexError”。

（14）执行下列程序段返回的结果是【　　】。

```
def f(* a): print(a)
q= [1,2,3,5,9]
f(* q)
```

A）(1,2,3,5,9)　　B）[1,2,3,5,9]　　C）SyntaxError　　D）TypeError

【答案】

A

【知识点】

详见 5.2 节。

【解析】

函数 f 的参数 * a 是一个可变数量的参数，实质上是一个元组，因此答案为 A。

## 4. 程序设计题

（1）定义 Max 函数返回一个数字列表中的最大值。

【答案】

```
def Max(arr):
    max_num=arr[0]
    for i in arr[1:]:
        if i>max_num:
            max_num=i
    return max_num
```

【知识点】

详见 5.1 节。

【解析】

定义一个变量 max_num 保存最大值，用 for 循环结构遍历列表，依次比较每个元素，使 max_num 始终保存大的一方。遍历列表后就可以得到列表的最大值。

（2）定义函数，测试输入的字符串是否为回文联（回文联即用回文形式写的对联，顺读倒读内容完全一样，如“山果花开花果山”）。

【答案】

```
def isHuiwen(str):
    if str==str[::-1]:
        print(str,'是回文联')
    else:
        print(str,'不是回文联')
```

【知识点】

详见 5.1 节。

【解析】

将字符串逆序排列并与原字符串比较，如果相同说明是回文联。用[::−1]方法获得逆序字符串。也可以用 reverse()方法实现。

（3）使用递归实现上题中的函数。

【答案】

```
def isHuiwen(str):
    if len(str)<2:
        print(str,'是回文联')
    if str[0]==str[-1]:
        return isHuiwen(str[1:-2])
    print( str,'不是回文联')
```

【知识点】

详见 5.5 节。

【解析】

根据回文联的特点，首先检查字串符的首尾字符是否相同。如果相同，缩小检查范围，去除字符串的首尾字符，检查新的字符串；如果不相同，则说明不是回文联。依此类推，如果字符串范围缩小到只剩 1 个字符时，说明是回文联。

（4）使用递归编写一个将十进制转换为二进制的函数（采用“除 2 取余”的方式，结果返回字符串形式）。

【答案】

```
def D2B(dec):
    result=''
    if dec:
        result=D2B(dec//2)
        return result+str(dec%2)
    else:
        return result
```

【知识点】

详见 5.5 节。

【解析】

十进制转换二进制时，该数除 2 取余直至商为 0，每次除法得到的余数转换为字符后添加到结果字符串中，最后得到的字符串就是对应的二进制数。以十进制数 10 为例：调用函数 d2b(10)，result＝D2B(5)的返回值。于是程序进入 D2B(5)的函数体(注意此时 D2B(10)的函数程序还没有执行完，正在等待 D2B(5)给它返回 result 的值)。在 D2B(5)的函数里，result＝D2B(2)的返回值，于是程序又进入 Dec2Bin(2)的函数里。依次类推，执行到 D2B(0)时执行 else 语句，返回 result，而此时 result 就是初始化时的空值，因此 D2B(0)返回空值给 D2B(1)，D2B(1)执行 return result＋str(dec％2)语句并将结果字符串“1”返回给 D2B(2)。依次类推，最终 D2B(10)获得了最终的二进制字符串。

(5) 有两种薪酬计算方案，请确定哪种薪酬方案更好。方案 1：每天 100 元。方案 2：第一天 100 元，第二天 200 元，第三天 400 元，每天的金额增长一倍，以此类推。使用名为 option1 和 option2 的函数计算在两种方案下的收入(假设工作 10 天)并输出结果。

【答案】

```
def option1(workdays):
    return 100*workdays
def option2(workdays):
    sum=0
    for i in range(0,workdays):
        sum=sum+100*2**i
    return sum
print('方案 1收入为：',option1(10))
print('方案 2收入为：',option2(10))
if option1(10)>option2(10):
    print('方案 1更好')
print('方案 2更好')
```

【知识点】

详见 5.5 节。

【解析】

方案 1 的收入计算很简单。方案 2 的收入根据题目要求：第一天的工资是 100 元，第二天增长 1 倍，即 100 * 2，第三天为(100 * 2) * 2＝100 * 2**2，…依此类推，第 n 天的收入是 100 * 2**(n－1)，将它们累加即总收入。哪个方案的收入多则认为该方案更好。

(6) 编写程序，要求输入一个人的姓名和目前的年薪，然后计算这个人下一年的薪水。如果年薪＜40000 元，则下一年的年薪将增长 5％。如果年薪≥40000 元收入，下一年的年薪除增加 2000 元外，还会增加超过 40000 元部分的 2％。使用函数处理输入、输出和计算新的薪水并输出，形式如下：

```
请输入姓名：**
请输入目前的年薪：****元
**新一年的年薪是：****元
```

【答案】

```
def calsalary(salary):
    if salary< 40000:
        salarynew= salary * 1.05
    else:
        salarynew= salary+ 2000+ (salary- 40000) * 0.02
    return salarynew
name= input('请输入姓名：')
salaryold= eval(input('请输入目前的年薪：'))
print(name,'新一年的年薪是：',calsalary(salaryold),'元')
```

【知识点】

详见 5.1 节。

【解析】

本题较简单，主要考察函数定义。根据题目要求设计相应函数，利用 if-else 结构分段计算两种年薪即可。

(7) 美国公务员退休制度中，一个人服务至少 20 年后可以在 55 岁退休。一个简单的计算退休金的方法如下：

① 计算收入最高的三年的年平均薪水，记为 ave。

② 计算月数/12，记为 yrs。

③ 计算比例：头 5 年 1.5％，接下来 5 年 1.75％，之后每年 2％，记为 perRate。

④ 取 perRate 和 80％中的较小值，记为 p。

⑤ 退休金金额为：p * ave。

编写一个程序，要求输入如下，并计算退休金金额。ave 和 p 的值应该通过函数来计算。

```
请输入您的年龄:**
请输入您的工作年限:**
请输入三个最高年薪中的第一个:****
请输入三个最高年薪中的第二个:****
请输入三个最高年薪中的第三个:****
您的退休年年薪是:****
```

【答案】

```
def salary_ave(salary_first,salary_second,salary_third):
    ave= (salary_first+ salary_second+ salary_third)/3
    return ave
def salary_percent(age,workyears):
    if workyears> = 20 and age> = 55:
        perRate= 5* 0.0175+ 5* 0.015+ (workyears- 10) * 0.02
        f= lambda x,y:x if x< y else y
        return f(perRate,0.8)      else:
        print('您还不能领取退休金')
age= eval(input('请输入您的年龄:'))
workyears= eval(input('请输入您的工作年限:'))
salary_first= eval(input('请输入三个最高年薪中的第一个:'))
salary_second= eval(input('请输入三个最高年薪中的第二个:'))
salary_third= eval(input('请输入三个最高年薪中的第三个:'))
ave= salary_ave(salary_first,salary_second,salary_third)
p= salary_percent(age,workyears)
print('您的退休年薪是:', p* ave)
```

【知识点】

详见 5.1 节。

【解析】

按照题目要求设计函数计算各种参数即可。这里注意一个限定条件,即工作年限满 20 年并且年龄满 55 周岁才可以领取退休金。程序中计算 p 时采用 lambda 表达式实现,这样使程序更为简洁,当然也可以用 if-else 结构。

(8) 英文中表示月份的 12 个单词中,如果包含字母 r,则称这个月份为 R 月。现有一个 Months.txt 文件,包含 12 行,每行有一个表示月份的单词。编写程序显示包含有字母 r 的月份。要求程序使用全局变量 months,并初始化为空列表。main 函数应该调用三个函数,一个用于使用文本文件中的内容填充 months 列表,一个用于从 months 列表中删除不包含字母 r 的月份,一个用于显示仍然留在列表中的月份名。

【答案】

```
def months_word(file):
    global months
    months=[]
    fp=open(file,'r')
    for line in fp:
      months.append(line.strip('\n'))
    #return months
    fp.close()
def months_delete(ls):
    for i in ls:
        if 'r' not in i:
            ls.remove(i)
def months_disp(ls):
    print(ls)
def main():
    file='Months.txt'
    months_word(file)
    months_delete(months)
    months_disp(months)
main()
```

【知识点】

详见 3.2 节和 5.1 节。

【解析】

函数 months_word()通过遍历文件对象,将 Months.txt 文件的每一行即月份放入列表 months 之中,同时用 strip()消除了每行的换行符"\n"。函数 months_delete()利用 remove()删除列表中不含"r"字母的月份。

(9) 威尔逊定理。如果一个数的因子只有 1 和它本身,这个数字即为素数。编写一个程序,使用定理"一个数字 n 是素数当且仅当(n−1)!+1 能被 n 整除"来判断一个数字是否是素数。程序应该定义一个返回布尔值的、名为 isPrime 的函数,并调用名为 factorial 的函数。

【答案】

```
n=eval(input("请输入大于 1 的整数:"))
if n<=1:
   n=eval(input("输入错误,请输入大于 1 的整数:"))
def factorial(n):
    if n==0:
        return 1
```

```
    else:
        return n* factorial(n-1)
def isPrime(n):
    if (factorial(n-1)+1)%n==0:
        return True
    else:
        return False
def main():
    if isPrime(n):
        print("该数是素数")
    else:
        print("该数不是素数")
main()
```

**【知识点】**

详见 5.1 节和 5.5 节。

**【解析】**

根据题目要求定义 factorial 函数，用递归函数实现阶乘。函数 isPrime()根据威尔逊定理判断输入的整数是否为素数。函数返回一个布尔值，如果是素数返回 True；否则返回 False。输入 n 时设定了一个判断条件，如果 n≤1 需要重新输入。

## 第 6 章习题答案及解析

### 1. 简答题

（1）简述什么是模块、如何把模块导入解释器，以及几种导入方法。

**【答案】**

在 Python 中，一个.py 文件就称为一个模块(Module)。模块最大的好处是大大提高了代码的可维护性和复用性。我们在编写程序的时候，也经常引用其他模块，包括 Python 内置的模块和来自第三方的模块。使用模块还可以避免函数名和变量名冲突。大体上有 import 和 from-import 两种形式。

（2）在解释器中如何终止程序并返回消息？

**【答案】**

Ctrl+C

### 2. 填空题

（1）面向对象程序设计的三要素分别为________、________、________。

**【答案】**

封装，继承，多态

（2）通过________语句可以导入模块。

【答案】

import

【解析】

import 还可以用于引入自己写的程序，if _ _name_ _=='main'的作用在此体现。如果直接执行 py 文件，模块名为_ _main_ _，程序会被执行；如果被当作模块引入到其他程序中，程序不会被执行。

（3）若一模块名为 m，________可以导入模块中的所有成员。

【答案】

from m import *

（4）在 Python 中，每个模块都会有自己的名称，可以通过特殊变量________查看；特别地，当一个模块被单独运行时，模块名称为________。

【答案】

_ _name_ _，_ _main_ _

### 3. 选择题

（1）同一类型的不同实例之间不具备【 】。

A）相同的操作集合　　B）相同的属性集合

C）相同的对象名　　D）不同的对象名

【答案】

C

（2）下列选项中，【 】不是 OOP 的基本特征。

A）类属性　　B）继承　　C）多态　　D）封装

【答案】

A

（3）下列程序段返回的结果是【 】。

```
class university:
    name="BJTU"
    age=120
p=university()
print(p.name)
```

A）BJTU　　B）"BJTU"　　C）120　　D）SyntaxError

【答案】

A

【解析】

p 是 university 类的一个实例，有两个属性，分别是 name 和 age。一般来说，不带括号的是属性，带括号的是方法。

（4）下列说法错误的是【　　】。

A）def 是定义方法的关键字

B）class 是定义类的关键字

C）类是对现实世界中一些事物的封装

D）方法是对现实世界中一些事物的封装

**【答案】**

D

（5）描述对象静态特征的数据元素是【　　】。

A）方法　　B）类型　　C）属性　　D）消息

**【答案】**

C

**【解析】**

方法是描述对象动态特征的数据元素，属性是描述对象静态特征的数据元素。

（6）在 Python 的类定义中，对成员变量的访问形式为【　　】。

A）＜对象＞.＜变量＞　　B）＜类名＞.＜变量＞

C）＜对象＞.方法（变量）　　D）＜类名＞.方法（变量）

**【答案】**

A

**【解析】**

一般来说，不带括号的是方法，带括号的是属性。访问对象属性时应该直接使用变量名而非类名。

（7）下列选项中【　　】不是面向对象方法的优点。

A）符合人们习惯的思维方法　　B）以功能分析为中心

C）代码复用率高　　D）更容易维护

**【答案】**

A

**【解析】**

对象是对现实世界的一种抽象；通常面向过程的方法更符合人们日常思考方式。

（8）当一个类定义了【　　】方法后，类实例化时会自动调用该方法。

A）auto()　　B）init()　　C）_ _auto_ _()　　D）_ _init_ _()

**【答案】**

D

**【解析】**

补充一点，把自己写的模块打包后，包目录应该要有__init__.py 文件，Python 把这个目录当成普通目录。__init__.py 本身就是一个模块，而它的模块名就是包名。

（9）有子类 China 和 Japan 继承了父类 Asia，若 c 和 j 分别是以上两个子类的实例，则 isinstance(c,Asia)、isinstance(j,Asia)、isinstance(c,China)、isinstance(j,China)返回的结果分别是【　　】。

A) True True False False　　B) True True True False

C) True False False True　　D) True True True True

**【答案】**

B

**【解析】**

子类的实例也是父类的实例。

(10) 面向对象程序设计着重于【　】的设计。

A) 对象　　B) 类　　C) 算法　　D) 数据

**【答案】**

B

**【解析】**

可以将现实世界中的对象经过抽象,映射为软件中的对象。对象在软件中是通过一种抽象数据类型来描述的,这种抽象数据类型称为类(Class)。一般情况下,面向对象程序都是由三个部分构成:类的声明、类的成员的实现和主函数。可见,在面向对象程序设计中,它着重于类的设计。类正是面向对象语言的基本程序模块,通过类的设计来完成实体的建模任务。

(11) 面向对象程序设计中,把对象的属性和行为组织在同一个模块内的机制叫做【　】。

A) 抽象　　B) 继承　　C) 封装　　D) 多态

**【答案】**

C

**【解析】**

面向对象就是将所有事物都抽象成类,用对象来调用类中的方法(也就是把客观事物封装成抽象的类,并且类可以把自己的数据和方法只让可信的类或者对象操作)。而类封装了属性和行为。属性(信息)用变量表示,行为(会做什么事)用函数表示。函数封装了逻辑代码。

### 4. 编程题

(1) 绘制一个五角星,注意填充为红色。

**【答案】**

```
import turtle as t
t.pendown()
t.color("red")
t.fillcolor("red")
t.begin_fill()
for i in range(5):
    t.forward(150)
t.left(144)
t.end_fill()
```

【解析】

画红色五角星需要画笔属性设置函数 color 和 fillcorlor，画笔运动命令 forward、backward 等，画笔运动方向改变命令 left、right 等，还有颜色填充函数 begin_fill 和 end_fill 命令。

(2) 绘制奥运五环图，其中五种颜色分别为蓝色、黑色、红色、黄色和绿色。注意根据实际效果调整圆形的大小和位置。

【答案】 略

【解析】

用 circle 函数画圆。用 pencolor 和 pensize 确定画笔颜色和画线宽度。用 penup 和 pendown 函数在画笔结束一个圆绘制，移动到其他地方绘制另外一个圆。画笔置于绘制起始点。

```
import turtle
x=-300
y=50
r=60
turtle.penup()

#第一个圈,蓝色
turtle.goto(x, y)
turtle.pendown()
turtle.pensize(15)
turtle.pencolor('blue')
turtle.circle(r)
turtle.penup()

#第二个圈,黑色
turtle.goto(x+2.5*r, y)
turtle.pendown()
turtle.pensize(15)
turtle.pencolor('black')
turtle.circle(r)
turtle.penup()

#第三个圈,红色
turtle.goto(x+2.5*r*2, y)
turtle.pendown()
turtle.pensize(15)
turtle.pencolor('red')
turtle.circle(r)
turtle.penup()
```

```
#第四个圈,黄色
turtle.goto(x+ (2.5* r) * 0.5, y- r)
turtle.pendown()
turtle.pensize(15)
turtle.pencolor('yellow')
turtle.circle(r)
turtle.penup()

#第五个圈,绿色
turtle.goto(x+ (2.5* r) * 0.5+ 2.5* r, y- r)
turtle.pendown()
turtle.pensize(15)
turtle.pencolor('green')
turtle.circle(r)
turtle.penup()
```

# 第 7 章综合训练题答案

1.

**【答案】**

(1) 形式不唯一,如 urn=[1]*83+[0]*58。

(2) sims 的作用是指定模拟次数,模拟次数越多,一般来说得出的精度越高。

(3) success 的作用是记录成功抽出两个白球的次数。

(4) 形式不唯一,如 success=[0]*sims。

(5) 作用是在壶中随机抽取两个球,1 表示白球,0 表示红球;draw 是 list 类型,可能是[1,1],[1,0],[0,1],[0,0]。

(6) 若 sum(draw)为 2,则表示抽到了两个白球,success 的第 i+1 个值会改变,否则不会改变。

(7) 若不做修改得出的结果是错误的,使用数学方法可以得出概率为 $P_{83}^2/P_{141}^2$,可以看出求出的概率比用数学方法计算得出的概率要大约一倍;错误在于 success 只是记录成功抽出两个白球的次数用的,不应该直接记录 sum(draw)的值,应当把 sum(draw)改成 1。

(8) 采用模拟的方式计算出的概率值具有随机性,不直接输出主要有以下两方面的考虑:①假使概率实际是 0.3,使用此方法计算出来的概率值几乎不可能为 0.3,但经过处理后可以使得出的概率值为 0.3;②求最终结果时往往不需要很高精度的概率,经过处理的概率值可以更为直观。round(np.mean(success),1)的作用是使用四舍五入的方式输出需要的概率值,实际应用中可以根据需要调整精度;不直接在格式控制中调整精度的原因是损失值较大.采用格式控制的方法调整概率值损失的数学期望是 0.5,而四舍五

入为 0.25，例如，概率实际为 0.3，算出的概率值为 0.29465，若使用格式控制的方法就会直接被调整为 0.2，而四舍五入为 0.3。

（9）题中程序求的是取出两个白球的概率。求取出至少一个白球的概率的程序形式不唯一，示例如下：

```
from random import sample
import numpy as np
urn= ________
sims= 100000
success= np.zeros(sims)
for i in range(sims):
    draw= sample(urn,2)
if sum(draw) != 0:
    success[i]= sum(draw)
print("取到至少一个是白球的概率{}".format(round(np.mean(success),1)))
```

（10）上面程序是不放回抽样的结果，有放回的程序不唯一，示例如下：

```
from random import sample
import numpy as np
urn= ________
sims= 100000
success= np.zeros(sims)
for i in range(sims):
    draw= [sample(urn,1) for j in range(2)]
if sum(draw)==2:
    success[i]= sum(draw)
print("取到两个都是白球的概率是{}".format(round(np.mean(success),1)))

from random import sample
import numpy as np
urn= ________
sims= 100000
success= np.zeros(sims)
for i in range(sims):
    draw= [sample(urn,1) for j in range(2)]
if sum(draw) != 0:
    success[i]= sum(draw)
print("取到至少一个是白球的概率{}".format(round(np.mean(success),1)))
```

2.

**【答案】**

（1）SciPy 的 stats 模块包含了大量的概率分布统计函数（原话是 This module

contains a large number of probability distributions as well as a growing library of statistical functions)；binom 的意思是服从二项分布的离散型随机变量(A binomial discrete random variable)；pmf 的意思是概率密度函数(probability mass function，注意，概率质量函数和概率密度函数不同之处在于：概率质量函数是对离散随机变量定义的，本身代表该值的概率；概率密度函数是对连续随机变量定义的，本身不是概率，只有对连续随机变量的概率密度函数在某区间内进行积分后才是概率)。

(2) 通过 753×800－50000X≥350000(X 代表出事故车辆数)可知，当 X 不超过 5 时符合题目要求，range(6)表示 X 分别为 0，1，2，3，4，5 的情况，753 表示车辆总数，0.006 为车辆出事故的概率。

(3) sum 接收的参数为题中 pmf 的返回值，pmf 返回的是一个数组，类型是 numpy.ndarray(即 n-dimensional array，意为多维数组，显然题中返回值为一维数组)，有 5 个元素，分别是出事故车辆为 0，1，2，3，4，5 时的概率，使用 sum 求和后即为出事故车辆不超过 5 时的概率。

(4) sum 的返回值是 0.70010170096167845(可能会因版本等原因而有些许差别)，但实际应用中并不需要这么高的精度，因此使用 round 函数控制精度；但直接使用得到的结果是 0.699999999999996，因此多加了一个 float 把 numpy.float64 转化成 float 类型。具体原因有兴趣的同学可以深入探究底层代码。

这里提供解决这道题的另一种思路，形式不唯一，示例如下：

```
from random import sample
L= [1] * 6+ [0] * 994          #表示概率 0.006
n= 0                           #表示符合条件的次数
sims= 100000                   #表示模拟的次数
for i in range(sims):
    k= 0                       #表示出事故车辆数
k+ = sum([sample(L,1)[0] for i in range(753)])
if k< = 5:                     #判断是否符合条件
    n+ = 1
print(n/sims)
```

**【解析】**

SciPy 库中的 integrate 和 stats 模块会是同学们最容易学习和应用的两个模块，分别对应着微积分和概率统计，SciPy 是以 numpy 为基础的，所以线性代数的知识(如矩阵)将会贯穿始终。https://docs.scipy.org/doc/scipy/reference/上有十分详细的官方教程，希望同学们养成自主学习的习惯。